MODELING AND SIMULATION TECHNIQUES FOR LARGE-SCALE COMMUNICATIONS MODELING

FINAL REPORT

OCTOBER 31, 1997

Sponsored by:
US Army Communications and Electronic Systems Command
Fort Monmouth, NJ 07703

Delivered in accordance with Contract Data
Requirements List data item B001
Contract DAAB07-97-C-D256

Contractor:
Steve Webb
809 Grayling Bay
Costa Mesa, CA 92626
714/957-1916

ABSTRACT

Modeling and simulation are together widely used throughout the Army, and vast amounts of computer time are used in running them. Of even more concern, however, is the quantity of analyst time involved in setting up and analyzing the results of the runs. Contributions that enhance the use of analyst time are therefore particularly welcomed. One aspect of efficient use is the confidence that users have in the ability of the simulation to represent the real world. To this end, an ongoing model validation effort was supported by developing and providing computer routines to calculate metrics that measure the degree to which simulation data match test data. Tests of random number generators were also developed and applied to CECOM models. Many techniques for speeding up simulation models rely on approximations that are adequate for the intended use of the models. In the case of engineering simulations, however, it is often desirable to maintain very high fidelity, even though it be superfluous for the current use, because future uses of the model may require it. For simulations that model random effects, a technique was found that is generally applicable, is easily implemented, does not compromise fidelity, and provides significant savings for making comparative studies with simulation models. Synchronization of the random number strings allows each entity modeled to have its own set of random draws for any combination of input parameters. If synchronization is in place, then statistical experiment design can also be used to provide information on the sensitivity of the output to input parameters. The report concludes with recommendations and an implementation plan.

FOREWORD

This is the final report covering research work accomplished under contract DAAB07-97-C-D256, entitled Modeling and Simulation Techniques for Large-Scale Communications Modeling, between the US Army Communications and Electronic Systems Command, Fort Monmouth, NJ, and Steve Webb, Costa Mesa, California, conducting business as a sole proprietor. This contract was awarded as a Phase I contract under the Department of Defense Small Business Innovation Research Program solicitation 96.2, Item A96-159. The period of performance was December 16, 1996 to September 30, 1997. The technical work was conducted by the contractor, Steve Webb, who wrote this report. Illustrations were done by Richard Escarcega. Technical monitor for CECOM was Phil Ciorciari of the RDEC Modeling and Simulation Branch, office symbol AMSEL-RD-C2-ST-S, telephone 732/427-2028. The assistance provided by John Wray of the Army Materiel Systems Analysis Activity, by William Ottly of the CECOM Modeling and Simulation Branch, and by Joseph Elmo of Atlantic Consulting Services, Inc. are also acknowledged. This report does not contain any potentially patentable subject matter.

TABLE OF CONTENTS

0. EXECUTIVE SUMMARY

Criticality of Modeling and Simulation Effort

Modeling and simulation are together a critical technology to the Army. Thousands of models have been written that are used for many purposes throughout almost all Army organizations.

Simulations are of many types. They are used to predict the effectiveness of weapon systems of the future as well as those in the field. They provide operational support in mission planning. The class of Distributed Interactive Simulations is used to mount war games that may involve many players simultaneously.

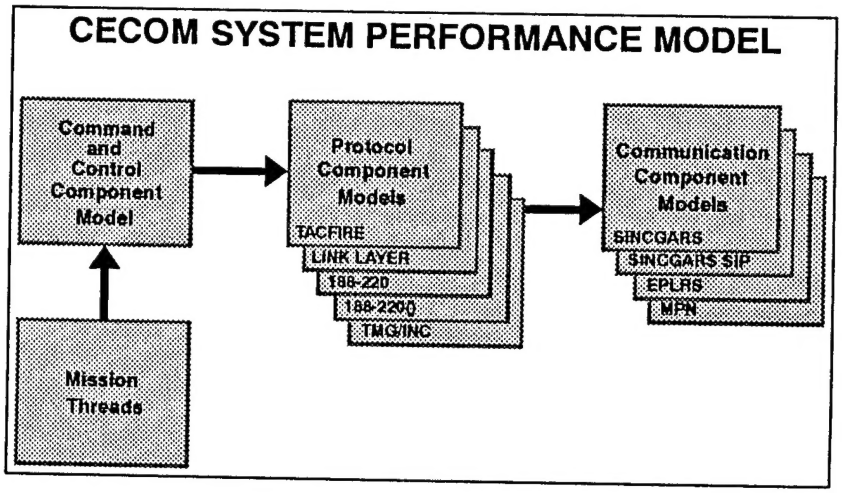
Of primary concern in this study is the class of engineering simulations as applied to digital communications systems. Such communication systems may be quite extensive and tend to contain a great many copies of elements that are essentially the same, such as radio sets. As engineering simulations, they are intended to contain faithful replicas of the physical aspects of the system. They often are built early in system design and are maintained to follow the changes made to the system as it is developed and fielded.

Since simulations run on computers, it would seem that minimizing computer time would be critical in reducing the vast total expense of the overall Army modeling and simulation effort. While this was true in the days of large centralized main-frame computers, it is relatively less important now that simulations are typically run on work stations under the control of the simulation user. Now the critical cost element is the analyst time involved in developing the simulation model, running it, verifying that the run was properly made, analyzing the output, and drawing conclusions. Computer time is generally only a marginal contributor.

Rather than considering the cost of an individual simulation run, it is more important to view it in the overall context of an analysis effort. If the effort can be structured more efficiently, so that the same conclusions can be reached in a shorter time or with a greater degree of confidence, then a real benefit is achieved.

Corresponding to this view, the scope of this study was made broad. While it included ways of decreasing the computer time of a simulation run, it also covered ways of using the simulation model and analyst together more efficiently. One way is to increase the confidence that others have in the validity of the simulation to aid the decision makers. This touches on the field of Verification, Validation, and Accreditation. The Army Materiel Systems Analysis Activity (AMSAA) was in the process of working with CECOM on VV&A of an important communications model. This provided an opportunity to cooperate in the area of techniques for assessing validation.

Validation Support for a CECOM Model



An important engineering simulation model is the CECOM System Performance Model (SPM). Nodes representing military operational facilities are linked by tactical communications networks using digital radios. Input message traffic may be scripted or generated statistically. Important parts of the model are the command and control functions, link protocols, the types of radios included, and the electromagnetic environment. The model can be used to study message completion rates and delays as a function of system load.

Validation is best done by a comparison of simulation output with real test data from the system being simulated. For this purpose an extensive set of development test data was available from the Enhanced Position Location Reporting System (EPLRS) radio engineering development effort. The question arose as to the best way to compare the data sets.

The standard statistical technique for comparing two data sets is the test of hypothesis. A null hypothesis of no difference between the sets is tested. If enough evidence is available to make the null hypothesis improbable, then the hypothesis is rejected; if there is not enough evidence, then all the procedure provides is an "I don't know" answer.

An alternative procedure was previously developed by this contractor, which provides a metric expressing how nearly the same are the two data sets. The quality of degree of sameness, rather than degree of difference, is the essence of validation. For the current effort, software routines were developed using this approach and provided to AMSAA through CECOM to support the data analysis for the validation.

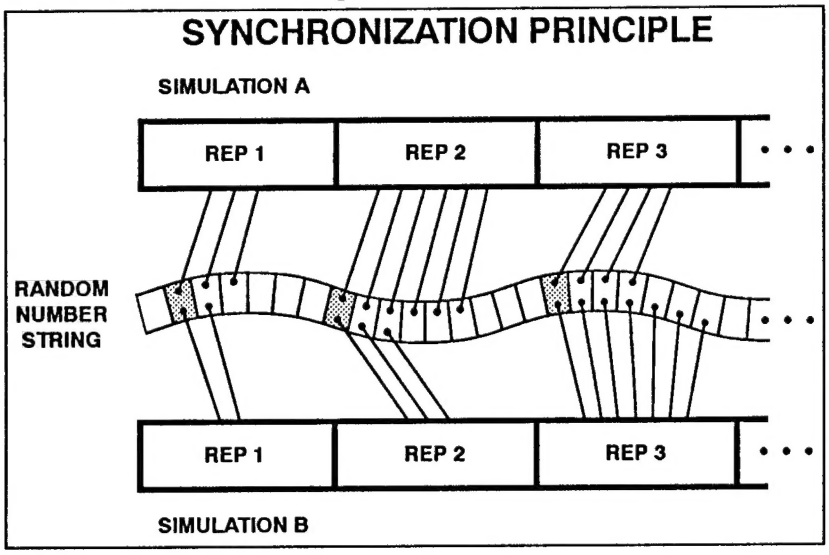
Like many simulations, SPM models some natural effects as random processes, which are mathematical abstractions that have been found useful for such purposes. The computer implementations of the random processes use an internal random number generator to model the outputs of these random processes. An examination of the SPM code shows on the order of 100 different calls to the random number generator. Such generators are really deterministic, but are supposed to generate sequences of numbers that share many of the properties associated with a truly random sequence. Unfortunately, some generators in use have been found to generate sequences with characteristics that are clearly not random. As an adjunct to the VV&A effort, a set of 5 test programs was constructed and tested on existing generators, both good and bad. They were then applied to the generator used in the SPM.

Synchronization of Random Number Draws

Simulations are often relied upon to make comparisons between different cases. These might represent different parameter settings, environmental effects, or different ways of using the system. The random number draws can introduce so much variability that the real effects of the factors to be compared are masked.

The masking effect of random variation is an aspect of the real world that such models simulate well. But unlike the real world, the "randomness" has an accessibility that can be exploited to increase the precision of comparisons. If exactly the same random number sequence is used on each side of the comparison, then much of the randomness is effectively cancelled out. Complications arise, however, if the change in parameters from one case to the other causes one or more extra draws to be made to the random sequence. The comparison will be tight before the first extra draw and loose after it. An observed difference may then depend more on when the extra draw was made than on the real difference introduced by the parameter change.

General techniques for ensuring that the random number draws remain synchronized were studied and refined. The procedures are easy to implement. In the simplest form the draws are made so that each replication has the same starting point (figure). Experiments were performed on a simplified communications model to evaluate the effectiveness of synchronization. It was possible to compare the sample sizes required with and without synchronization to obtain an equivalent precision in a comparison of simulation outputs. It was found that for this model synchronization provided precision equivalent to an increase in sample size by a factor ranging from 2 to over 100. The benefit is then seen to be large if the effect of interest is small compared with experimental error.



Another technique for making comparisons using simulations is provided by the introduction of statistical experiment design techniques. This has the promise of allowing large numbers of comparisons to be made with efficient use of computer resources. Setting up, running, and analyzing large numbers of simulation cases is time consuming for the analyst, however. A system is envisioned to aid the analyst in carrying out the process. It would allow a baseline simulation to be the center point for a detailed investigation of parameter effects. Next, single-variable sensitivities would be determined, then the general effects of all the parameters when several parameters are simultaneously varied. Test cases illustrating this approach were developed.

Major Findings

1) Engineering simulations are difficult to speed up in general ways because good ways of eliminating computation may often compromise model fidelity. Fidelity of an engineering model should be maintained because the model may have different uses in the future for which apparently superfluous fidelity is necessary (further discussion is on page 10).

2) Formal validation in which simulation results are compared with test results from the system simulated is an important means of increasing confidence in simulation results. The validation metrics developed have proven to be useful indications of the degree of similarity between simulation and test (p 17).

3) The random number generation scheme used in the SPM is adequate for the ways that it is used in the model (p 22).

4) There is no provision in SPM to keep the random number draws synchronized. Because there are many draws made throughout the model for many purposes, two cases with different parameters would not be expected to use the same random numbers for the same purposes. The safest thing to do is to make all runs with different starting seeds. SPM has a provision for doing just that (p 18).

5) Synchronizing random numbers so that the same strings can be used for making comparisons has a major benefit for comparative studies using a simulation model (p 31).

Major Recommendations

1) Synchronize the random number draws in SPM and other important models used for comparative studies (p 32).

2) Until a communication model can be synchronized, make all runs using scripted message input, rather than traffic statistically generated at the time the run is made. The scripted input may be generated statistically offline before the simulation run, and saved in case any comparative runs are to be made in the future (p 42).

3) Until it is synchronized, make all runs using different seeds. This is a normal mode of operation in SPM using the low order bits of the system clock to provide the first seed (p 42).

4) To support future execution time improvements, introduce timing instrumentation into SPM by calling the system clock before and after major parts of the code are executed (p 42).

5) Consider improving the random number generator from the current adequate one to a good one; GEN_K or GEN_H introduced later are candidates, subject to further testing (p 42).

6) Consider the introduction of a semi-automated system for generating, running, and analyzing statistical experiment designs to study system performance as a function of input parameters (p 39).

1. INTRODUCTION

The Importance of Simulation in Defense

The subject of modeling and simulation has been identified by the DoD as one of twenty technologies critical to ensuring the long-term superiority of weapon systems [Schuppe 1991]. It was categorized as an enabling technology that offers capability for advances in weapon systems. It is widely used for analyses ranging from advanced planning to operational support.

Some simulation models are deterministic, always giving the same result for the same input values. The majority of models, however, attempt to reflect the real-world variation that we refer to as "random." One might argue that all variation could be explained by a sufficiently detailed model. It will suffice to define random as a mathematical abstraction; a random variable or process produces outputs that individually and jointly follow specified probability distributions that reflect the variation seen empirically. The implementation in a simulation is usually accomplished by using so-called pseudo random number generators. Typically the generator is a small piece of computer code that from an initial *seed* generates a sequence of real numbers between zero and one. Although the sequence is deterministic given the seed, the sequence is intended to have many of the properties that a truly random sequence would have. These include

- Uniformity across the interval 0 to 1
- Apparent independence from one number to the next
- Long cycle length before the sequence repeats.

When a random number generator is used, in order to understand the behavior of the model we need to run the model more. This may be done by simulating longer periods of time, by looking at more than one sample, or by a combination. By running the simulation for a number of independent replications (using a different string of random numbers each time) we gain insight into how variable the responses of the system may be. By averaging results, we are able to estimate with greater precision what the average response of the system will be. There is a tendency in simulation studies to underestimate the number of replications needed to develop a full understanding of the variability of the responses.

Often a model is used to make comparisons of results using different combinations of values of the input parameters. For example, a gun model might vary muzzle velocity, bullet mass, and shape. As a model is first used, the large effects are soon discovered and understood and more interest centers on effects that are small compared with the random variability. Also of interest are how different combinations of system inputs jointly affect the system outputs. Both these desires may lead to making more simulation runs. Once again, the number of cases run to gain a systematic understanding of how the system responds over all its possible input conditions tends to be underestimated.

All in all, many models are in existence that are frequently used. Together they use a vast amount of computer time. The need to understand variability better and to average out its influence, and the need to understand the effects of changes in combinations of variables contribute pressure to make even more simulation runs.

Types of Simulation Models

Even if the scope of simulation models is restricted to the DoD, there are really many different types, each with its own characteristics. There is no generally accepted categorization, and many simulation models have characteristics of several basic types. Some of the more important types for Army applications will be reviewed based on those given in a catalog of about 525 models compiled by the Joint Staff [1992].

Many models are constructed to estimate the effectiveness of a weapon system. They range in complexity from simple models with a high degree of aggregation used in the early planning phase for a system that might be developed sometime in the future. At the other extreme are highly detailed models for systems under development. Models also range in scope from mission level models such as Osprey [Webb et al 1987], which treats a complete negation mission for many resident and replacement military satellites in low earth orbits, to time-stepped models of the flight of a single round against a target.

Phenomenological models treat such things as the propagation medium of a signal or bullet, weather, interference, compatibility, physical terrain, and the like. They frequently appear as parts of other models. Examples are TIREM [ECAC 1983], GT-sig, which models thermal backgrounds, and MAPS, a CECOM model for electronic warfare studies. Often numerical integration is a key part of the computational load.

A similar class contains models for lethality and survivability. They contain detailed descriptions of the geometry of the target and weapon, treat phenomenology of weapon interaction with elements of the target, and often must consider multiple shotline effects as the interaction proceeds. Again they may be used in other models. They can be computationally intensive. An example is SQuASH, a detailed point-burst lethality code developed by Ballistics Research Laboratory for artillery against armored vehicles.

Models built for training and education must respond in the manner and time scale of the real system as it is perceived to those using it. Often accuracy is not as important as an interface that is similar to the real human interface. An example is TACSIM, which produces realistic intelligence reports (at the SCI level), but admittedly compromises sensor resolution.

Operational support models are developed later as a system is deployed and used. They may evolve from earlier system effectiveness models. They are used to check the effects of anticipated actions in a specific scenario. Many examples occur in the fields of logistics (PROLOGUE evaluates logistics aspects of operational planning at the theater level) and communications (JTIDSC2 estimates the performance of a JTIDS network as deployed).

There is a great deal of current interest in distributed interactive simulations (DIS) for support of training and exercise rehearsal. A simulation often involves many participants, each with their own computer and role to play. The simulation runs on each of the computers with information exchanged between them by means of protocol data units. The volume and speed of data exchange are issues that are much more critical than simulation computational speed. Examples are the Untethered Land Warrior [Sauerborn 1995] or Janus [Army 1986].

Possible Techniques for Accelerating Simulations

Since computer simulation is so prevalent, the amount of computer time used to produce simulation results is enormous. Techniques for decreasing computer run time are therefore of considerable interest. A reasonable step is to see if the numerical calculations done in a simulation can be arranged more efficiently. This might work very well at first, but has diminishing returns. Optimizing compilers have been around for a long time, and do a good job in rearranging computations for best utilizing the capabilities of existing computers. Improving them is difficult.

Another approach is to do the calculations faster, perhaps with a newer computer or an advanced architecture that, for example, might allow parallel computation. New computers are interesting, but costly to buy, install, integrate, learn, debug, and get running existing application software.

An alternate approach is to look beyond the computational level and find a completely different solution that gives equivalent answers. A perhaps apocryphal story concerns Gauss as a young boy. To keep him occupied during math class, his teacher tasked him to add the integers from 1 to 100. After apparently doodling for a minute or so, he came up with the answer. He showed the astonished professor that by adding the first and last numbers, second and next to last, etc, always getting the same result, he could reduce the problem to a single multiplication.

An example from this writer's past arose in the late 1960's at McDonnell Douglas Aerospace. A library of subroutines for missile guidance system analysis contained a subroutine for solving simple linear equations that had been "optimized" by specializing to three dimensions and simplifying data indexing. A numerical analysis specialist observed, however, that the solution algorithm used was Cramer's rule (ratio of determinants). Use of Gaussian elimination easily beat the "optimum" subroutine.

Another example arose when converting a system-level antisatellite simulation program from large main-frame computers to run on a small personal computer. Although only a small part of the code, the engagement planning function uses the large majority of the run time. A simple prescreening test was added to the code to avoid even trying to compute an engagement plan if the satellite track was beyond the maximum reach of the interceptor missile. This simple technique reduced run times by a factor of eight.

Other standard approaches are to use variable step sizes in numerical integration procedures, to perform detailed calculations offline and replace them in the simulation model by either a table lookup or by an analytic approximation, or to aggregate low order results. In fact, aggregation has been called the most pervasive type of approximation in simulations [Bratley 1983].

So rearranging the calculations, better numerical analysis, or prescreening are valid approaches. However, there does not seem to be a general scheme for finding them. The challenge is to do it in a way that does not degrade fidelity.

The Cost of Making a Simulation Run

Just a few years ago computer systems were built around large expensive main-frame computers. Simulation runs were made in batch mode by sending a "job" to the mainframe often using smaller computers to perform input and output to the faster mainframes. Costs of acquiring, equipment leasing, maintaining, and operating the computer centers were recovered by charging users fees based primarily on the amount of computer time used.

Now it is commonplace for simulations to be run on work stations that are purchased as capital equipment and operated and maintained by the user. The cost of computer time has become highly nonlinear. The following table makes a whimsical analysis of the benefit of getting a simulation to run twice as fast.

From	To	Value	Reason
1 sec	½ sec	None	Response is essentially instantaneous anyway
16 sec	8 sec	Appreciable	A computer pause while the user is ready to make the next input is very annoying
3 min	1½ min	Little	Either delay is long enough to get a cup of coffee or go to the restroom
90 min	45 min	Large	Analysts put in long runs just before they go to meetings; if runs are shorter, then there will be pressure to get the meeting over with
20 hr	10 hr	Small	In either case there is one turn around per day
4 days	2 days	Very great	Just one turn around in a work week causes everyone to forget what the project is about; two or even three makes for good discussion in the weekly report.

Although this analysis lacks the rigor for serious consideration by a human factors journal, it does illustrate that the real benefit of a time saving is in what the human operator can do with it. It almost goes without saying that time saving is much more important on a model for which individual runs are long, even if the total time involved is smaller than for other models that are run much more frequently.

Often more important than computer time is the time taken by an analyst to set up, verify, and analyze the results of the runs. This suggests that a broader look at run effectiveness is preferred to a limited treatment of just computer run time.

Organization of This Report

Section 2 is a short review of large-scale communications models in terms of their distinguishing characteristics and use at CECOM. An introduction to the model selected for special emphasis, called SPM, is given.

If its users have confidence in a simulation model, then there will be a general perception that time used in running and working with the model will be well spent. Validation is an important contributor to such confidence. An ongoing validation effort of the SPM was supported as part of the current study. A new statistical approach to comparing simulated and real data was reduced to computer routines that were then used for the validation. What was done is discussed in Section 3. Mathematical details are discussed in Appendix A.

Many simulations use random numbers, and most of these rely on routines supplied with the computer system. Historically, some such routines have been disappointing in their emulation of theoretical random properties. Tests for random number generators were developed, themselves tested on 14 generators of both good and bad quality, and then used to assess the generator in SPM. A general description of this work and its application to the SPM generator are given in Section 4. Appendix B gives details of the tests, generators, and starting seeds used. Fortran listings of the tests and generators are also included. A more comprehensive collection of testing results is given in Appendix C.

The work on synchronization of random number generators is given in Section 5. This includes illustrations of what happens when synchronization is lost, a simple communication simulation built to test synchronization schemes, and an assessment of the gains obtained using it for comparative studies. The section contains a recipe for implementing it and a refinement as used in a large simulation is given in Appendix D.

The topic of statistical experiment design as it can be applied to simulation studies is discussed in Section 6. Included in the section is a detailed illustration of its use. Issues involved in implementing an automated approach are discussed. A catalog of actual designs useful for simulation studies is given as Appendix E. Details of the example application are in Appendix F.

Section 7 reviews other general techniques for speeding up simulations that were looked at. Most general techniques involve approximations that might degrade fidelity if not used carefully. Because of the importance of high fidelity for engineering simulations, these techniques were de-emphasized in this study. Some work was done on a model for studying a technique called staged aggregation. Results, which were largely negative, are presented in Appendix G.

Section 8 presents conclusions and discusses how improvements might be implemented.

References are collected at the end of the report.

2. CECOM LARGE-SCALE COMMUNICATIONS MODELS

General Features

As the Army moves into the 21st century, increasing emphasis is being placed on the power of information. CECOM supports this evolution towards Force XXI by developing digital information systems designed to promote rapid and accurate decision making. Modeling and simulation are used by CECOM as effective and efficient means of supporting the development, integration, and testing of these systems.

This work was supported by the Modeling and Simulation Branch of the C3I Modeling and Simulation Division of the CECOM Research, Development, and Engineering Center. The concern of this branch is engineering models that support ongoing CECOM development areas, primarily in the area of digital combat radio networks.

The philosophical basis for engineering models is such that many of the standard ways of speeding up the models may be unpalatable. These models are often started early in the concept definition phase of a system acquisition and follow it through design, development, testing, and deployment. The uses for the simulation model are not all anticipated ahead of time. High fidelity is usually prized. Often the model will emulate the actual workings of the system, particularly those parts that are implemented as computer code. While a systems analysis model might represent communications protocol by a statistical delay, an engineering model would be expected to contain a detailed and faithful emulation of the protocol.

As a result, the use of approximations or simplifications in parts of the model may not be acceptable. While they might give satisfactory results for the large majority of applications, some future application of the model might give misleading results.

For example, consider a hypothetical model of an infrared seeker that is used to select an aimpoint from an extended target image at the terminal phase of intercept. A satisfactory and fast model for use in high fidelity system effectiveness simulations might use a geometric representation of the target focal plane image that depends on engagement approach angles, together with statistical characterization of miss distances in each dimension. An engineering model, on the other hand, would be much more likely to represent pixel-by-pixel processing and to emulate the real-time algorithm used to calculate an aim point. For most applications both models would give essentially equivalent statistical results. If anomalous behavior were observed during a system test, however, the engineering model would be much more useful in studying possible ways of avoiding similar problems in future tests.

CECOM System Performance Model

This model (SPM) was selected as a prototype for study and analysis in this effort. It is an engineering simulation of the performance of combat radio networks connecting Operational Facilities. These might be a Brigade Tactical Operations Center, a Battery Fire Direction Center, an individual soldier acting as a forward observer, or an armored cavalry vehicle. Messages might include intelligence reports, node status and location, weather predictions, movement orders, firing plans, fire support requests, target assignments, etc.

The message traffic between the nodes may be scripted or may be randomly generated in the course of the run. In either case, additional message transmissions may result from the receipt of messages.

The major parts of the SPM are:

- Command and Control Component Model

- Protocol Component Models, including models of the Tacfire, Link Layer, 188-220, 188-220(), and TMG/INC protocols.

- Communications Component Models, including Single Channel Ground and Airborne Radio System (SINCGARS), SINCGARS System Improvement Program, Enhanced Position Location Reporting System, Mobile Subscriber Equipment Packet Network (MPN), and earlier AN/PRC- radios.

- Communications Realism Submodel, including propagation effects due to terrain between transmission and reception points along paths taken by vehicle-borne radios.

The SPM is implemented in the General Simulation System language. This system was developed and is marketed by Prediction Systems, Inc. of Spring Lake, New Jersey. It allows the model to be described in a structured english language based system. Subroutines or functions written in C code may be used. It runs on Silicon Graphics workstations under the UNIX operating system.

A typical run might involve 200 nodes interconnected by 30 networks each containing 10 to 30 radios. A single node may be on several (up to 7) networks by using multiple radio sets. A time period of several hours might be simulated, with the simulation run itself taking several hours. Important outputs are the percent of messages that are completed, the percent that are completed within a specified speed of service, and the individual completion times (from message creation to receipt).

3. SIMULATION MODEL VALIDATION

Importance of Validation

The scope of the current study has been expanded somewhat from speed of running a single simulation to the broader context of using the model more effectively. If a model is not valid for its intended use, then effort is wasted. If it is validated and accredited, then the resulting increase in confidence contributes to effectiveness.

Recent definitions have been developed by a Senior Advisory Group convened by the Military Operations Research Society [Ritchie 1992]:

> **Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.**
>
> **Accreditation: An official determination that a model is acceptable for a specific purpose.**

By these definitions, validation is a continuing effort that might be going on throughout the entire sequence of conceptualizing the simulation model through exercising it in a production mode. The validation work might culminate in several accreditation decisions based on different purposes for the model.

In both military and civilian contexts simulation often provides a major input for making critical decisions concerning the real system being simulated. Those persons who are responsible for making such decisions are rightly concerned about the fidelity of the simulations in representing the real systems. The simulations must correspond faithfully to those aspects of the real system that contribute to making the right decision. From this point of view, the subject of validation of the simulations is the basis for the simulation being an effective and trustworthy representation of the real system.

As one might expect, a great deal of attention has been given to this issue. However, there has not been a great deal of resolution. It is generally agreed that in order to validate a simulation model, empirical data are necessary and statistical procedures are desirable. However, omnibus methods for validation do not exist, and many approaches are problem specific. This concern led the Army Research Laboratory to sponsor a research study conducted by this contractor. The main thrust of the work was the development of a metric that expresses the degree of validation that has been demonstrated. In its basic form, the metric is designed to compare empirical data from the real system, assumed to be a very short list, with data from the simulation, which may be extensive since the simulation is by its very purpose intended to be an inexpensive surrogate for the real system.

Hypothesis Tests and an Alternative Approach

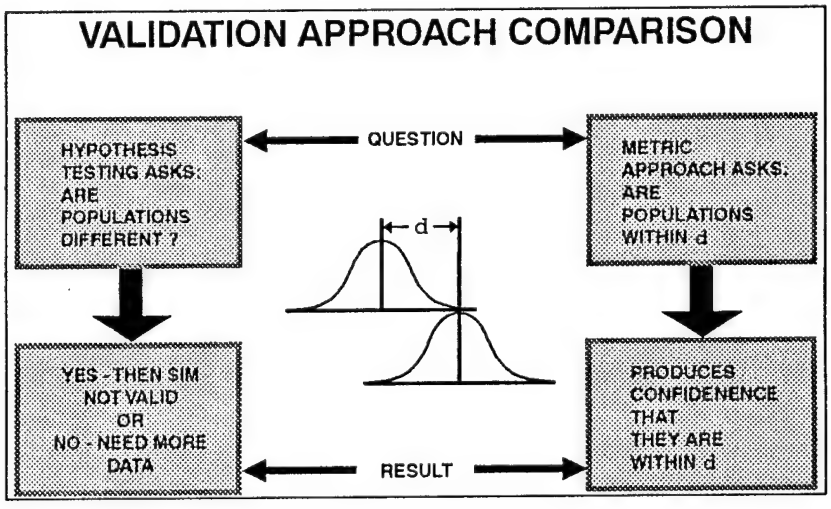
It is agreed that validation should be based on a match between empirical and simulated data, but there is not usually any clear definition given for what constitutes agreement. Because variation is an essence of empirical reality, a statistical approach seems appropriate for assessing the closeness of the agreement. Many texts and journal articles propose standard statistical tests.

Formal statistical hypothesis tests are designed to determine whether there is sufficient evidence to assert that two populations are different; for example, is a new treatment better than a control. If they are really different, then as more and more data are obtained, the chances are better and better that the measured results will show them to be different. No matter how small the real difference is, if the sample sizes are large enough, the null hypothesis of no difference will eventually be rejected.

For validation the question is whether or not two entities are essentially the same. This is more than a semantic distinction; almost any null hypothesis of no difference between two populations can be rejected if enough data are available. We already know that a simulation differs from its target system: one runs on the computer and the other in the real world. What we would like to know about a simulation is whether it gives results that are close enough to those obtained with the real system. Let us suppose that it is possible to establish criteria of closeness for each of the outputs of the system that are of interest. We can then consider the degree of faith that the simulation gives results that are close to those from the real system.

This formulation is similar to a confidence limit problem. A confidence region is usually defined for fixed confidence coefficient γ to be the set of parameter values Θ that are such that a test would not be rejected. Here the set of values of the difference between simulation and reality is given, and we may use the confidence level as a metric for validation. This metric ranges from 1, indicating perfect agreement, down to 0, a mismatch. If a lot of real data are available and the model matches these data well, the metric should be close to one. If data are few, or the model does not match the data well, the metric should be near zero.

To state this idea another way, if the simulation in fact matches the real system well, then as more and more data are obtained the metric should increase to near one. If the agreement is poor, then it should decrease to zero. If the situation is borderline, then the confidence would likely just dither around intermediate values.



Validation of the SPM

A Verification, Validation and Accreditation effort for the CECOM System Performance Model involving CECOM and AMSAA was underway, which provided an opportunity to support the VV&A analysis under this contract. Initial discussions of the philosophy of validation of simulation models led to a decision to try out the metric approach rather than standard hypothesis testing. Previous efforts of a similar nature using large quantities of available data had experienced the problems noted with statistical hypothesis testing. In particular, a standard test (the KS test) of whether two populations have the same statistical distribution had rejected the hypothesis that simulated and observed message delay time data were from the same population, even though it was clear that differences were of no practical consequence.

The validation plan for the System Performance Model centers around a field experiment involving the Enhanced Position Location Reporting System (EPLRS). For over a hundred needlines (links between nodes), data are available on the number of messages sent, the number received, the number received within the specified speed of service, and the transmission delay time. The simulation was set up with the identical laydown of radio sets and scripted with the same set of messages sent.

It was decided that as part of the support to the VV&A effort, computer routines would be developed to implement the computation of the validation metric. Previously the calculations had been done by hand, which was impractical for the large data sets expected. It was further noted that the data for comparison were of two types:

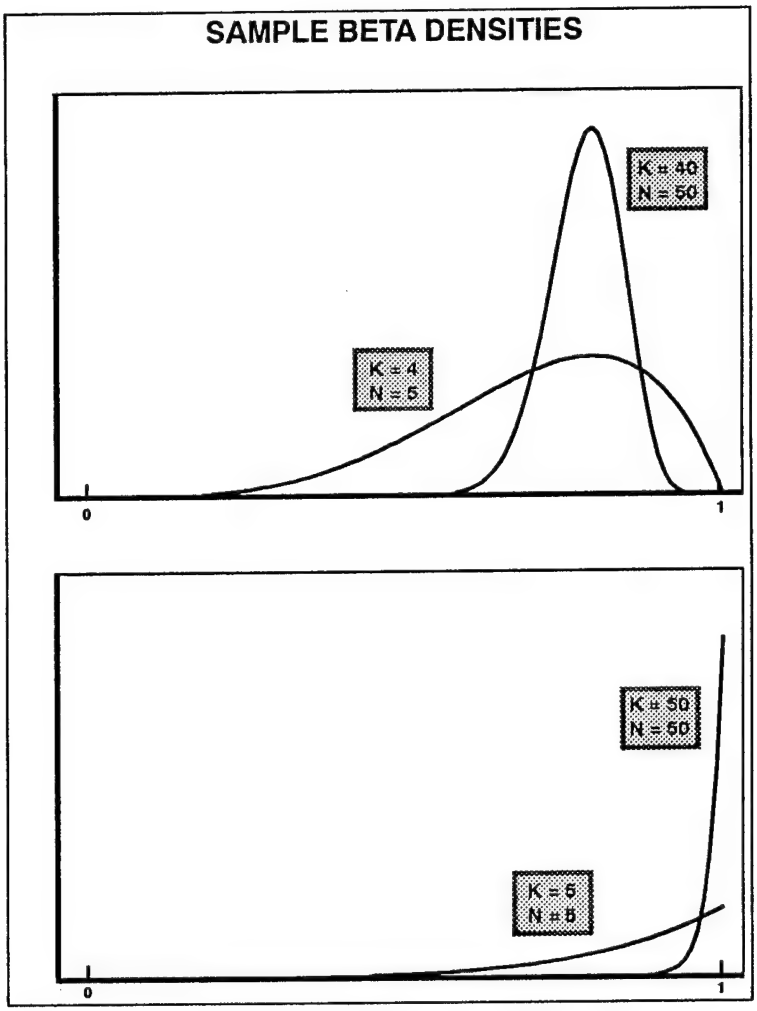
- **Binomial data** in which the result is one of two possible outcomes. Each attempt at message transmission either met with success or not. If successful, the delay was either within the speed of service requirement or not.

- **Delay time data** representing the actual time from transmission until reception.

Statistical procedures are available for testing hypotheses about whether two binomial proportions are equal, or for establishing confidence intervals on single binomial proportions. Methods do not seem to be readily available, however, for determining the confidence with which two proportions lie within a specified interval. A new procedure for this purpose was required. Moreover, the procedure needed to work for sample sizes ranging from just a few (say 5) to several thousand. The case in which all trials were successes was common, but for some a significant percentage were failures.

The most familiar statistical techniques for numerical data assume a functional form for the distribution, such as the Gaussian or exponential. For the delay time data there was no basis for any particular form from either experience or theory. Therefore a nonparametric approach, in which no particular form is assumed, seemed appropriate.

A Metric for Binomial Data



If the unknown probability of success p on any one trial is viewed as having a probability distribution which is uniform between zero and one before data are obtained, then after observing K successes in N trials, p has a beta distribution. The figures show that the probability density is diffuse for small values of N and concentrated for larger values (the vertical axes are scaled arbitrarily to emphasize the relative shapes). The peak is near the ratio K/N. If K = N, the mode is at the extreme right end of the density.

Using this approach, the probability that two binomial proportions p_1 and p_2 are within $\pm d$ of each other can be calculated from the beta distributions on each p. The numerical evaluation of this expression over wide ranges of the five parameters (d, N_1, K_1, N_2, and K_2) is challenging, but is required for the SPM data. Integrals of the beta densities are replaced by summations with finite step sizes. The step sizes must be small enough to give satisfactory accuracy, but not so small as to give unacceptably large computation times. If N is small, a fairly large integration step size will give accurate results. If N is large, however, the density is very peaked and a small step must be used. On the other hand, the density is negligible over part of the range. A second problem is the exact evaluation of the binomial coefficients for N things taken K at a time, which is equal to N! / K! (N–K)!. If N is large then this expression will cause computational overflow except when K is close to N or 0. The techniques used to balance accuracy and speed are given in Appendix A.

The question arises why all the concern with accuracy. Surely the user would not care if the confidence were really .86 when .88 is reported. The answer is to insure that data sets will be internally consistent. For example, the same result should be computed if the samples are reversed, or if the roles of K and N - K are reversed for both samples. No jumps should be discernable as sets of data values make transitions through boundaries separating regions in which different computational procedures are used. Because all such constraints may not be anticipated, it was decided to strive for three decimal place accuracy in the computer implementation, called BETALIM2, that was supplied for use in the VV&A effort.

A Metric for Delay Time Data

For processing the message delay time data, a second procedure has been developed. The metric expresses the confidence that the two populations are within a given tolerance $\pm$ d, and is constructed based on differences between the ordered observations from the simulation X_1, X_2, ... , X_m, and the ordered observations from the real system Y_1, Y_2, ... , Y_n. The theory is based on the presumption that if the two are the same, then any arrangement of X's and Y's in the combined sample is equally likely.

The procedure developed uses the Mann-Whitney U statistic, which is a count of the number of instances in which a member of the second sample is less than a member of the first. The value of U can range from 0 if all the observations in the second sample are greater than any in the first, to $N_1 \times N_2$ if all are less than any in the first. If two samples are very different, then U will have a value close to one of these extremes. If they are the same, then U will probably have a central value. If a sample of N_1 X's is really from the same population as a second sample of N_2 Y's, then if all the $N_1 + N_2$ observations are sorted, then any particular pattern of X's and Y's is equally likely to occur. The probability distribution of U in this case can be computed from a recursion relationship giving the number of possible arrangements of N_1 X values and N_2 Y values that give the same value for U. For large values of N_1 and N_2 a Gaussian approximation is available. This is based on the asymptotic distribution, but is considered to be "reasonably" accurate for equal sample sizes as small as 6.

To form a metric giving the confidence that two samples represent populations that are within an indifference $\pm$ d of each other in location parameter, the U statistic is computed twice using the second sample values with d added and subtracted. The values of U are compared with the percentage points of its distribution to obtain values that are differenced to form the final metric.

The routine is complicated by the need to treat cases where one or more differences $X_i - Y_j$ are exactly equal to the tolerance $-d$ or $+d$. Different combinations of possibilities need to be treated correctly in the computerized algorithm.

A Fortran routine was developed implementing this procedure. At first, only the large-sample approximation was implemented. The working version, called METRIC8, was developed that improves the large-scale approximation slightly, but more importantly adds the exact computation for cases in which both sample sizes are less than 20. This routine works by building a table of the exact distribution, which is referenced for particular values of U. This version was used by AMSAA to reduce the data for the validation effort.

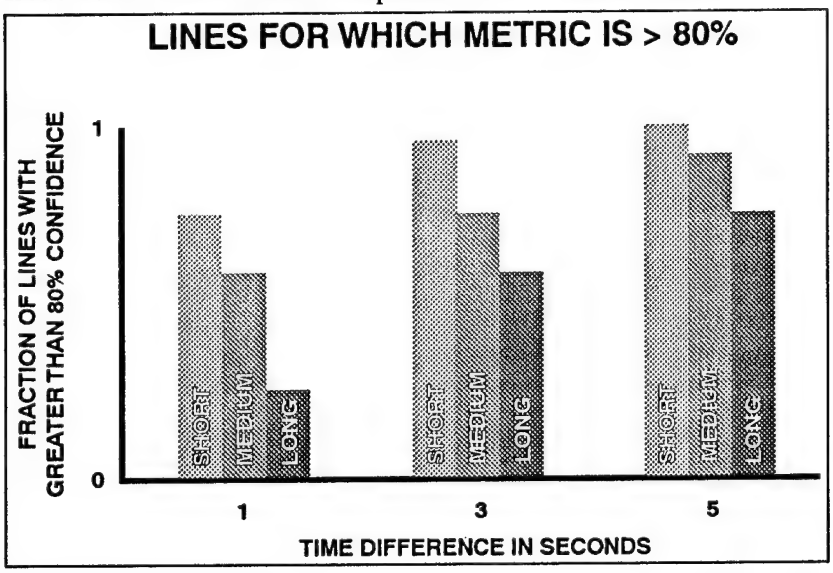
Attempts were made to develop a version that would use the recursion relationships to obtain distribution values as they are needed. If successful, this approach could have covered the cases when only one sample size is less than 20. The attempts failed to achieve results in reasonable amounts of computer time because they got tangled up in a complex tree of procedure calls, so the approach was abandoned. This case occurs rarely if at all in the SPM data.

Application of the Approach to SPM

The metrics from the software routines provided were used to support the comparison of the simulation based on test data obtained from EPLRS development testing. The results of the application were presented by John Wray [1997] of AMSAA. He contrasted the conventional statistical approach using statistical hypothesis tests to the metric approach, which provides the difference level of the model in comparison to the test data.

Two scenarios using low data rates were compared, containing 132 radios and 115 duplex needlines. Approximately 10% of the radios were located on moving vehicles during the test. The simulation was run using the scripted communication traffic and the actual radio positions and movement from the tests. For each scenario, the simulation was replicated three times using different random number seeds.

For the message delivery time data, comparisons were made using the value of the difference d from 1 through 5 seconds. Plots were obtained giving for each time difference the percentage of needlines for which the confidence as indicated by the metric was at least 80% and 90%. Similar plots were obtained for the message completion rate, using difference levels of from 5% to 10% of the completion rates from the tests.



A further comparison was made in which data were separated by length. Intelligence and Electronic Warfare messages exemplified long messages, for which the required speed of service was 90 seconds; Fire Support messages represented medium length messages, which were required in around 20 seconds; short messages, for which the requirement was 4 seconds, were Air Defense Artillery messages. It was found that for long messages, the model tended to predict slightly shorter delivery times than found in the test data, and for short messages the opposite trend was observed. The figure is representative of those used by Wray to present results (data in the figure are not real).

In summary, the metric approach appears to give comparisons of simulation and real data that are useful for analysts and that contribute to confidence in the simulation models.

4. RANDOM NUMBER GENERATION

Use of Generators in SPM

SPM makes many calls to its pseudo random number generator. From a review of the code there are around 100 calls made in over 60 of the 300 or so procedures that comprise the model. According to the code and the available documentation, these are used for many purposes, including the following:

- Statistical message generation
- Statistical treatment of mutual interference
- Message priorities
- Scheduled time slots for each radio set
- Frequency resource for each radio set
- Random access and processing delays
- Random occurrence of collision
- Percentage of chips affected by each collision
- Occurrence of frame synchronization
- Number of hops in a frequency segment
- Transmit profile for an interferer
- Occurrence of a bit error
- Selection of time slot block index
- Whether each leg of a relay is established
- Calculation of instantaneous power

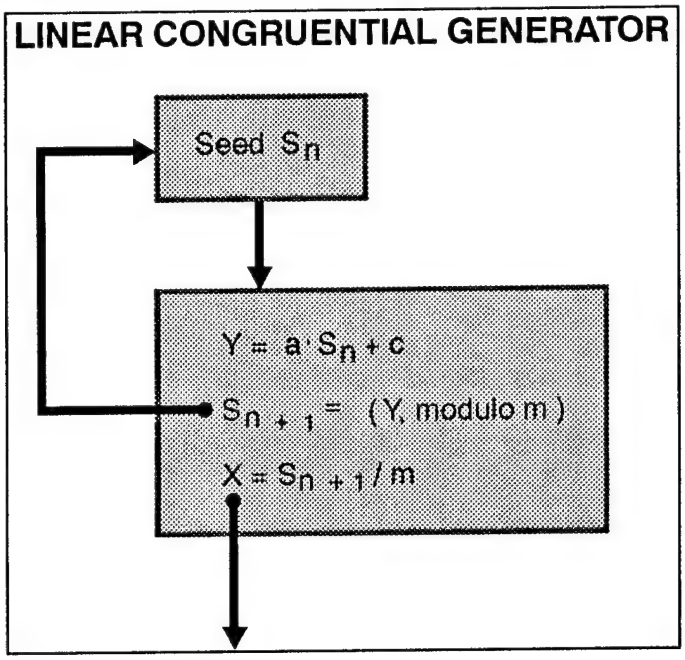
There are several options available in GSS for generating random numbers: 1) using RANDOM as a variable name sets that variable to the next random number in the interval 0 to 1; 2) calling UNIFORM produces a random number between specified limits; 3) calling EXPON gives an exponentially distributed number, useful as a message generation time; 4) calling TEXPON gives an exponentially distributed number truncated to avoid excessively long values; 5) NORMAL gives a number with the Gaussian or normal distribution; 6) TNORMAL gives a truncated Gaussian number.

A random number generator must have a starting point and SPM has an interesting feature to ensure that the starting point is random from one run to another. This is to use the current value of the low order bits from the system clock. Optionally, the user may set the starting point to a specified value read from one of many input files. However, this requires a change to a line of code and recompilation. This procedure would be the norm during the debugging process when a new case is set up or code changes are made.

It is recognized that generators are really deterministic; they always give the same sequence given the same starting point. However, their effective use depends on their behaving like random sequences in terms of frequency distribution, lack of repetition, and apparent independence of successive elements. Testing for these properties is the next topic of discussion.

Properties of Random Number Generators

Surprisingly, the random number generators supplied in specific computer systems have historically often been deficient. Due to their importance in CECOM simulations, an effort in evaluating the generators used seems worthwhile.



Most generators work as follows: an integer seed (usually a large integer) is either supplied as an argument when the routine is called or stored internally in the code; the next integer in the random number string is calculated and used to replace the seed; the routine returns a real number between 0 and 1 based on the new value of the seed. The most common type is shown in the figure. Its properties are determined by the multiplier a, the increment c (often just 0), and the modulus m. Since the cycle will eventually repeat and the period can be no larger than the modulus, m is usually taken to be a large value, equal to or close to the word length of the computer.

There has been a great deal of theoretical work on good choices for the multiplier, increment, and modulus. Much of it relies on number theory to determine choices that make the cycle length as long as possible, and then to prevent obvious lack of randomness given maximum cycle length. A summary is given by Knuth [1969].

The linear congruential generator calculates the n-th value from just the previous value. A more complex type computes the next value from two of the previous values, often with a lag between. For example, $X_{n+1} = (X_n + X_{n-k})$ modulo m. These are called Fibonacci generators because they generalize the Fibonacci sequence 1, 1, 2, 3, 5, 8, 13, ... in which each number is the sum of the preceding two. They also require internal storage of intermediate numbers.

In addition to long cycle length, a good random number generator would share other properties with a truly random sequence. Among these are:

- Uniformity between 0 and 1
- Uniformity of pairs over the unit square, triples over the unit cube, etc.
- Independence of successive values.

Although some theoretical results are available, it is usual to rely on empirical tests of such properties.

Specific Tests Developed

Although packages of tests for random number generators probably exist, none was available for use in this effort, so a battery of tests was developed and programmed. Each will be described.

The first test simply prints out the first three draws for each seed in both integer and real form for inspection. The second attempts to check cycle length by brute force. It makes up to 30,000 successive draws of integers for a given starting seed, compares each with up to 20,000 last entries in the list, and stops if a match is found. It was found to be impractical to use.

Tests 3 through 7 were found to be effective tests for generators. They are fast running and effective in separating known bad from good generators. Tests 3 through 6 implement ideas mentioned by Knuth [1969]. The seventh is an implementation of a concept for a more stringent test due to Marsaglia [1985].

Rantest3 – Pair Uniformity Makes 20,000 draws of pairs of real numbers and sorts them into a 64×64 grid. It then does a chi-square test to check for uniformity.

Rantest4 – Gap Test Counts the number of successive real number draws for which no element lies between values α and β. Proceeds for a total of 7000 cases. It was run with the width of the gap ($\beta - \alpha$) equal to .1 and the beginning point set to the five values 0, .15, .45, .75, and .9. Thus, five different gaps were used in the testing. This is a particularly important test for applications such as SPM because random number draws are often used to determine probabilities of events. Good performance means that the recurrence times of such events will be as expected by theory.

Rantest5 – Permutation Test Draws 5000 sets of 6 numbers and categorizes each by which of the 6! possible permutations they fall in when sorting them by size.

Rantest6 – Run Test Finds the lengths of runs in which successive values are all increasing, for 40,000 runs. Runs of 8 or more are lumped together.

Rantest7 – Overlapping Triples Sorts 30,000 overlapping triples of numbers into an $8 \times 8 \times 8$ grid, and tests for uniformity. Because the triples are overlapping, the test also will detect lack of independence.

In order to test the tests, a battery of 14 random number generators has been assembled. Three are generators that have been used in developing simulations by the author. The rest are implementations of generators mentioned as both good and bad examples by other authors. The generators are described and listings given in Appendix B. Testing of each Rantest was performed with a set of 23 seeds. Most tests take about a minute of computer time on a 486 personal computer. The exception is the cycle length check in Rantest2, which takes about 25 minutes per seed if no repeat is found. Fortunately, other tests seem to detect generator-seed combinations for which short cycle length is a problem.

Output from the Rantest Routines

Suppose Rantest3, which counts the frequencies of successive pairs of random numbers, is applied to a specific generator. The result is a 64 by 64 array of integers, whose sum is 20,000. If the generator works properly, then all cells are equally probable and the numbers in the cells would show some random variation around the average of $20000 / 64^2 = 4.88$. Let E_i be this expected cell count and let A_i be the actual cell count for cell i. Then the statistic $(A_i - E_i)^2 / E_i$ summed over all cells has a distribution that is approximately chi-square with degrees of freedom equal to the number of cells less 1. The test routine calculates this sum and uses the functional form of the chi-square distribution to calculate the probability that the value obtained would be this large or larger due to chance if the cells are really equiprobable. A partial output for GEN_K follows (the complete output is in Appendix C).

```
PAIR UNIFORMITY CHI-SQUARE TEST FOR GENERATOR: GEN_K
   20000 PAIRS, SORTED INTO  64 BY  64 GRID
SEED        VALUE CHISQUARE   P VALUE
   2            1   4129.54   .349621
   3            2   4088.58   .526107
   4            3   4016.49   .806821
   5            4   4088.17   .527910
   6    123456789   4120.52   .387005
   7      1111111   3944.40   .952956
   8         6999   4119.30   .392197
   9        65536   4159.03   .238767
     ...
```

If the generator is good (this one is), then the P values will themselves be randomly scattered between 0 and 1. If the generator is not good, then at least some of the P values will usually be small. Some are so extreme that the P values are near 0:

```
PAIR UNIFORMITY CHI-SQUARE TEST FOR GENERATOR: GEN_F
   20000 PAIRS, SORTED INTO  64 BY  64 GRID
SEED        VALUE CHISQUARE   P VALUE
   2            1 450315.00   .000000
   3            2 449300.40   .000000
   4            3 450047.90   .000000
   5            4 449123.90   .000000
   6    123456789 451006.80   .000000
   7      1111111 449488.40   .000000
   8         6999 448922.00   .000000
   9        65536 449632.20   .000000
     ...
```

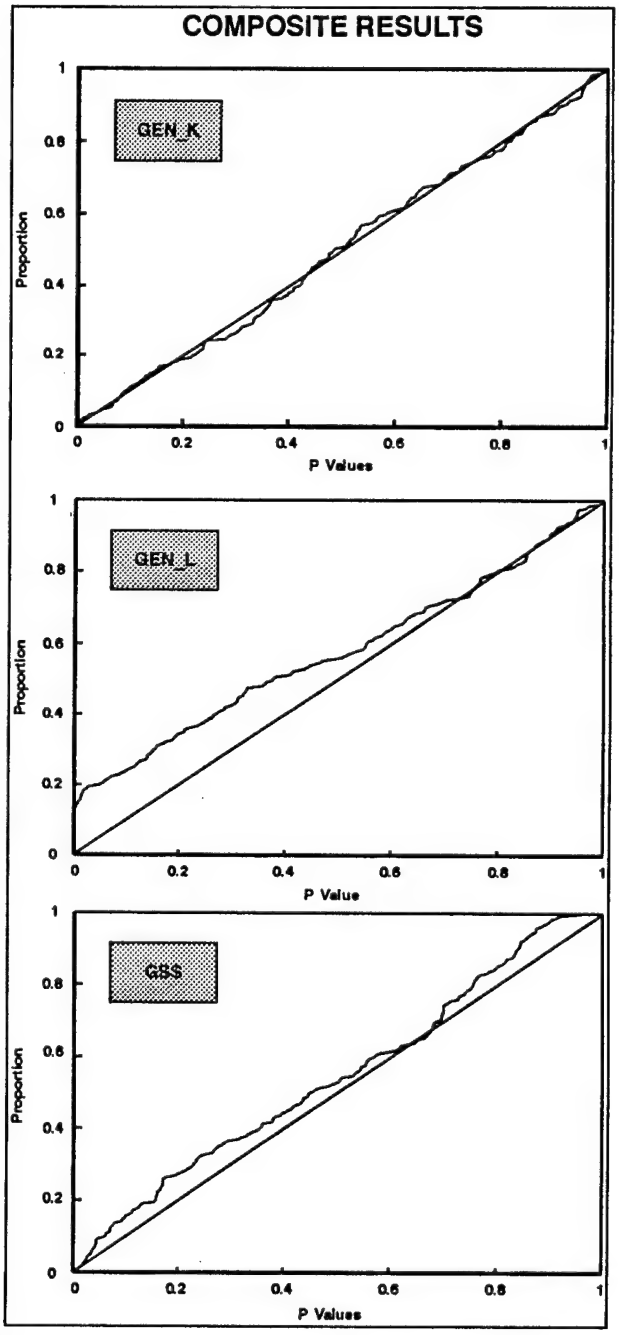
Output tables from the other Rantest routines are similar, except that Rantest4 is written to give P values for 5 different gaps, so that 5 columns of P values are printed.

Testing the SPM Generator

For applying the Rantests to the SPM generator, a slightly different procedure was used than for the test generators. The testing procedure for the 14 test generators was to compile the Rantest routines with the call changed to the name of each generator in turn. The generator used in SPM is the one provided by the GSS system. For testing it, a file of 70,000 successive values was constructed for each of the 23 seeds, which was then used to run the tests at CECOM. Rantest3, Rantest5, and Rantest7 each use a fixed number of random numbers less than 70,000. Rantest4 uses a variable number until a specified gap count is achieved. The gap count was reduced from 7000 to 6500 so that the file would not be exceeded. Similarly Rantest6 was modified; 40,000 runs of length up to 8 was reduced to 25,000 runs of length up to 6 for the GSS generator.

For summarizing the results of all tests, it is convenient to use a plot. The P values resulting from all tests on a given generator, approximately 200 in number, are placed in rank order, then the values plotted against order number. A good generator like GEN_K gives a plot that is essentially a straight line from (0,0) to (1,1) as shown in the top figure. The second figure, for GEN_L, uses the multiplier of the notorious RANDU. This generator was supplied by IBM as a library routine in the 1960's. It was good enough to be supplied but was discovered by users to have decidedly nonrandom properties. It deviates considerably from the 45 degree line.

The final figure gives results for the GSS generator. The deviation from the 45 degree line is noticeable, but not as great as for GEN_L. This would suggest that the generator is not the best, but is probably adequate for most applications. In particular, the applications in the SPM would not seem to require a great degree of subtlety in the interrelationships between successive random number draws to attain answers that are reliable for the needs of the analyses made.



And What if a Generator is Bad?

If a given generator fails one or more of the Rantests, then the sequence of generated numbers does not share some property of theoretically random numbers. How serious this may be depends a lot on how the numbers are used. Marsaglia [1985] developed his tests to support "... increasingly sophisticated Monte Carlo uses, such as in geometric probability, combinatorics, estimating distribution functions, comparing statistical procedures, generating and testing for large primes for use in encryption schemes, and the like."

One misuse of random numbers is to rely too heavily on low-order bits. If entities are to be assigned at random to two classes, one might be tempted to make the assignment according to whether a random integer is odd or even. The low order bits are known to be decidedly nonrandom even for generators that may otherwise be reasonably good. The Rantests did not check low order bits.

Marsaglia goes on to say that for most purposes generators work remarkably well, and even bad generators may be good enough.

In most cases the user of a bad generator will not notice that anything is amiss. Attempts were made to use bad generators in simulation models discussed elsewhere in this report to see if their effect would be noticed, with negative results. The output looked much the same with good or bad generators. Presumably it would be possible to find some element of the behavior of the simulation that is similar to a Rantest that a bad generator flunked, then introduce additional model output that would find this bad behavior, but doing so would be difficult and contrived.

Still, if a generator is bad, then the simulation using it is not modeling exactly what was intended. The answers are wrong without giving the user any indication. A Monte Carlo routine designed to evaluate an integral will be converging to the wrong value. We may speculate that the degree of error may not be very serious in a case of interest, but whether or not this is true is unknown.

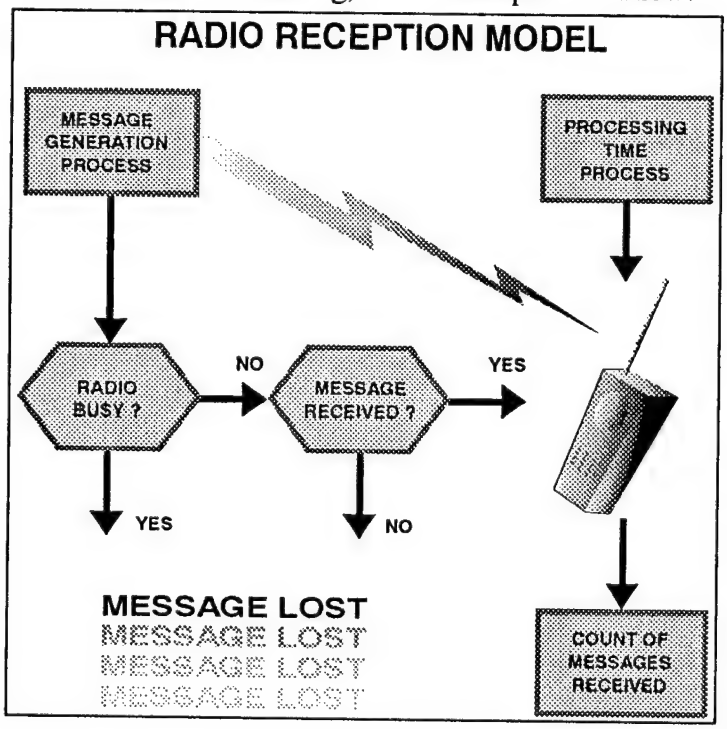
On the other hand, it is known that bad random number generators can lead to bizarre behavior observable in details of the output. A specific example of this happening could not be found, however. More often, an analyst may try to explain strange details as being an artifact of the generator (the author has been guilty).

As a final thought, suppose that the generator has been demonstrated to be beyond reproach. Then any concerns that the analyst might have are alleviated. Any strange-looking behavior is either a problem with the model or else is just due to a strange sequence of random numbers. It is easy to determine which is the case, simply by rerunning the case with a different random seed. For example, a simulation of communications between four radios, to be discussed in the next section, had the first 5 transmission attempts all interrupted by interference, which should be a rare event. In this case it was just an unusual sequence of draws. The effect disappeared as other cases were run.

5. RANDOM NUMBER SYNCHRONIZATION

Purpose of Synchronization

Often simulation runs are made to compare different system options. The original requirement description for this effort mentions use of simulations to evaluate the impact of doctrinal or operational changes, and the effectiveness of technology insertion. In order to make effective comparisons between different cases, it is desirable to cut down on variability of the comparison. One way to do this effectively, is to take care that each entity that uses random numbers will always see the same stream. If this is not done, then the random numbers used in different cases may start out the same, then diverge at some point. This can make comparisons between cases misleading, as an example will show.



Suppose that a radio set is used to receive messages that are sent at an average rate of one every 3 time units. When a message arrives, there is a probability of 10% that it will be missed. If not, then the radio set is occupied in receiving and processing the message for a random time of average length 5 units.

A simulation was written to determine the number of messages received in 100 time units with this setup. This program consisted of about 400 lines of Fortran, of which only about 20% was new, the rest being reused from another model. First, all random number draws were made from the same string. The average was about 12 messages, with the range being 8 to 16.

A second run was made in which the arrival rate was increased to one message every 2½ time units. More messages should be received. Out of 20 cases, only 10 were such that more were received. Five received the same and 5 received fewer. The reason is not that somehow a form of contention arises at a higher input rate, but that different random numbers get associated with the messages.

The simulation was rewritten to segregate the draws for message arrival times, reception probabilities, and processing times. This is done by keeping track of three different seeds and making each call with the appropriate one. This time, a comparison showed that in 15 of 20 cases there were more messages received at the higher rate, in 4 the same number, and in only one case one fewer messages was received. The sample sizes used are not large enough to prove the case, but the results suggest that random number synchronization bears a closer look.

A Simulation for Developing Synchronization Methods

It is possible to structure random number draws in such a way to reduce the variability of comparisons to be made using a simulation model. To demonstrate the use of this technique and to study its effectiveness in communication systems simulations, a simulation model was developed, which is simple but of more substance than that on the previous page. Once again, it was adapted from an existing event-oriented model. Of about 900 lines of Fortran, about a third were reused, consisting of the event calendar, input and output, and utility routines; two-thirds were new, consisting of the processing associated with the occurrence of the events.

A small number of observation posts are able to communicate with each other by radioing messages of varied lengths. Events occur that cause one post to send another a message. However, the message transmission may only be initiated during a time slot that is assigned to the sending post. The assignment of time slots cycles among the posts.

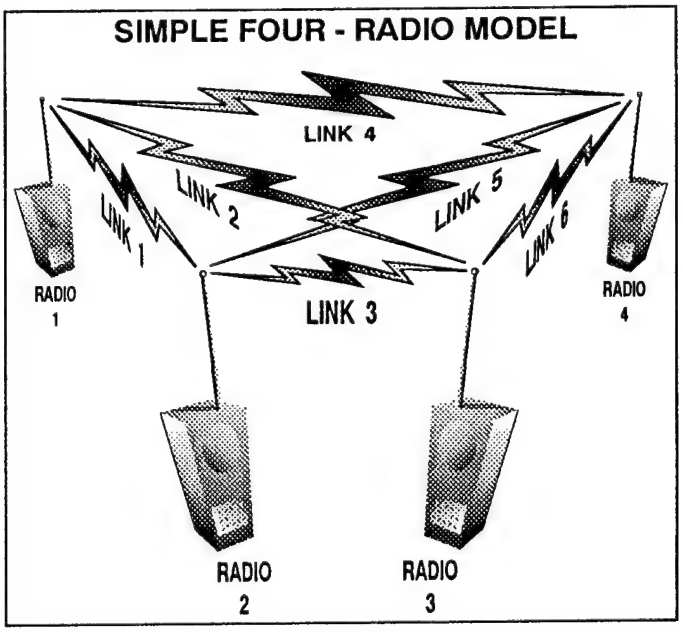
If a transmission attempt is made when the intended recipient is busy sending or receiving a message with another post, then the message is placed in a storage buffer. Similarly, if a message triggering event occurs while the post is engaged in communication, the new message is placed in a buffer. Attempts are made regularly to send the messages in the buffers as the assigned time slots occur.

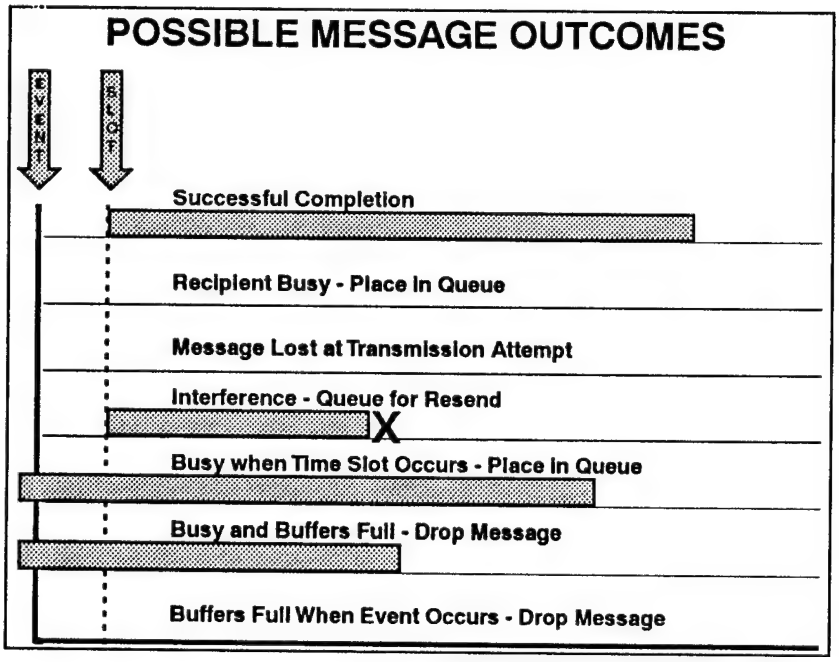
There are a fixed number of storage buffers available in each radio. If a message triggering event occurs when the buffers are all in use, the corresponding message is dropped.

If a transmission attempt is made when the recipient is not busy, then with high probability the message will go through. There is a small probability that the message will not be completed, however, and the sender is unaware that the message is not sent.

If the message goes through successfully, then the sending and receiving radios are busy for the duration of the transmission. However, at random times interference events occur on the links between pairs of posts. If an interference event occurs during the time that a message is being transmitted, then the message goes into the storage buffer system for later retransmission. An interference event has no effect if the link on which it occurs is not in use at the instant of its occurrence.

The system operates for a fixed period of time. At the end of this period, no more message triggering events occur, but transmissions continue until all buffers in each radio have been cleared.





Possible Outcomes of a Trigger Event

When a triggering event occurs it is associated with a particular radio. If that radio is not busy, then in effect the corresponding message is composed and placed in an "immediate send" buffer for transmission when the next time slot comes up for the radio. At that time, an attempt is made to send the message. If the recipient is not busy, then the transmission is initiated and with a little luck goes through to a successful completion (figure).

If the intended recipient is busy at the time the transmission is attempted, then the message is placed in a storage buffer for later transmission. With a certain low probability (nominal 2%) the transmission attempt will fail and the message be lost.

Random interference events occur on each of the links. If such an event occurs while a transmission is underway, then the transmission is interrupted. The message is placed in a storage queue for later retransmission. The retransmission will start again at the beginning of the message, with no credit given for the part that was previously transmitted.

If the radio is busy when the triggering event occurs, then the message is placed in the storage queue. Similarly, if the radio was not busy and the message was composed in the send buffer, but before the radio's time slot comes up the radio becomes busy by receiving a transmission from another radio, then the message is removed to the storage queue. Both these options are lumped in the fifth line of the figure.

If a message is to be placed in the storage queue for any of the above reasons, but the queue buffers are all full, then the message is lost. The last two lines of the figure illustrate that this may happen whether or not the radio is busy.

If a radio is not busy but has messages stored in its buffers, then as each of its assigned time slots comes up, it moves the message from the first storage buffer into the send buffer for the next attempt. All other messages are moved up in the queue. If the attempt is not successful, then the message goes back to the end of the queue.

Sample Timeline

A sample timeline using the model is shown in the table. An interference event that occurs when a link is not in use is denoted "Interference link y," where y is the link number, while interference occurring during a transmission is denoted "Interrupt Mx," where x is the message number. As a standard case, the model is run with 4 radios and the following parameters:

- Mean interarrival times of trigger events: 20 time units for each radio
- Mean service time: 6.5 time units for each radio
- Number of buffers: 4 in each radio
- Probability of message loss: .02
- Assigned time slot length: .1 time units for each radio
- Mean interference interarrival time: 30 time units.

Time	Radio 1	Radio 2	Radio 3	Radio 4
0.00	idle	idle	idle	idle
0.18	Interference link 2		Interference link 2	
0.75	Event 1			
0.80	Start transmit M1	Receive M1		
2.29		Event 2 queued		
2.81		Event 3 queued		
4.08	Interference link 2		Interference link 2	
5.04	Interrupt M1	Interrupt M1		
5.20	Start transmit M1	Receive M1		
5.31	Event 4 queued			
6.39			Event 5	
6.60			Try M5 – busy	
8.35			Interference link 6	Interference link 6
9.63				Event 6
9.90			Receive M6	Start transmit M6
11.52				Event 7 queued
13.97			Event 8 queued	
18.48			Interrupt M6	Interrupt M6
18.99	Interrupt M1	Interrupt M1		
19.00	Receive M8		Start transmit M8	
19.58	Event 9 queued			
20.48	Interrupt M8		Interrupt M8	
20.50		Start transmit M2	Receive M2	
21.10	Receive M7			Start transmit M7
21.98		Interrupt M2	Interrupt M2	
22.10		Start transmit M3	Receive M3	
22.94	Event 10 queued			
23.47	Event 11 dropped			
24.12	Interference link 1	Interference link 1		
25.75	Complete M7			Complete M7
29.71		Complete M3	Complete M3	
29.80		Receive M5	Start transmit M5	
32.92		Complete M5	Complete M5	
33.00	Receive M8		Start transmit M8	
33.98	Interference link 1	Interference link 1		
...	...	...	...	...

Output from the Model

Responses for the model are the total number of messages formulated, the number dropped, the number lost, the number transmitted successfully, the average length of time from message trigger until transmission is completed successfully, the number of messages for which this time length is less than a specified speed of service threshold (set to 50 time units), and the excess time after the operational period ends until all messages are disposed. Derived responses are the proportion of total messages received successfully and the proportion that are received within the speed of service threshold.

The model is typically run for several replications and the means and standard deviations of the responses tabulated.

Results averaged over 20 replications of a 200 time unit period are as follows:

- Total messages (triggering events) = 39.35 (±1.10)
- Number dropped because of buffer saturation = 0.80 (±.21)
- Number lost because transmission attempt failed = 1.10 (±.25)
- Number completed = 37.45 (±.97)
- Number completed within 50 time units = 32.30 (±1.05)
- Average completion time of those completed = 26.11 time units (±2.07)
- Average percent completed = 95.34% (±.76)
- Average percent completed within 50 time units = 82.60% (±2.50).

In addition to the average value, the standard error of the mean is given for each response. The standard error of the mean is the standard deviation of the response from the 20 runs, divided by $\sqrt{20}$. This measures the variability of the average from the 20 replications, and can be used to compare results from different cases.

Some modeling details will be mentioned. The message triggering event process is Markov; that is, times between successive arrivals are independent and exponentially distributed. The mean interarrival time for each radio is specified as input. The interference event process is also Markov, with the same mean time between occurrences for each link. The message transmission lengths are Erlang distributed, with the shape parameter equal to 3, but with the means possibly different for each radio.

Although the model was generally run with 4 radios, it is dimensioned for up to 49. The corresponding number of links between pairs of radios is 1176.

An Attempted Comparison and Synchronization

The simulation was run a second time with the number of buffers increased from 4 to 5, to evaluate the improvement in the average percent completed. The result, however, shows that the percent completion decreased to 94.51 and the average number dropped increased to 1.25. How can increasing the number of buffers decrease performance?

A closer examination of the results from the two cases shows that the first replication is identical, and the second (from which the timeline given on page 27 was extracted) is identical up until time 23.47, at which time message 11 is not dropped. After that time, radio 1 has an extra message cycling in its buffers as its time slots come up. Not until time 97.2 is the message sequence altered, when radio 1 sends a different message with an earlier completion time. Shortly thereafter, a difference in messages sent causes a conflict so that one less message initiation occurs, one less draw is made, and the arrival processes then start to diverge for all radios. Eventually a sequence of close event arrivals all for Radio 1 occurs, causing several messages to be dropped even with the larger number of buffers. Ultimately, 6 messages are dropped for this replicate, as opposed to the nominal case for which only message 11 is dropped.

The simulation was rewritten so that the random number draws will remain synchronized. The sequence of calls to the random number generator is

- Initialization:
  - first message trigger event time for each radio
  - first interference event time for each link
- At message trigger event
  - next trigger event time for this radio
  - recipient for the message
  - length of the message
- At message transmission event
  - draw against probability of message loss
- At interference event
  - next interference event for this link.

The synchronization is effected by introducing an array of seeds for the random number draws. There are four seeds associated with each radio, one for each different usage, and one with each link. At the start of each replication, the array is initialized with a separate random number generator, with its own control seed, that will always provide the same starting point independent of the input parameter values. The basic generator used is a standard published routine called UNIF2 in Appendix B and the initialization generator is called GEN_A. The resulting model may be called Syncsim and compared with Plainsim, the original.

Comparison of Syncsim with Plainsim

Comparable results for the baseline case with the two simulations are in the table. Since the purpose of this example is to compare results with and without synchronization, these results were obtained by trying ten different seeds and using those providing the best match for these responses. The agreement between the nominal Plainsim and nominal Syncsim is therefore somewhat closer than what would be seen between randomly chosen samples.

RESPONSE	PLAINSIM	SYNCSIM
Number of messages	39.35	39.45
Average time	26.11	25.75
Completion ratio	95.34%	94.98%
On-time ratio	82.60%	82.04%

Of interest is what happens when the number of buffers is increase to 5 in Syncsim. The completion ratio increases from 94.98% to 96.42% and the average number dropped decreases from 1.50 to .75 messages, which follows intuition and common sense. Cases were run with each of the key input parameters increased and decreased from nominal, and all responses now vary in the expected direction. A good comparison between a nominal case and an excursion is provided by the differences in responses between each replication. For example, the completion percentages for the 4 versus 5 buffer case and their differences follow.

	PLAINSIM Completion Percentage			SYNCSIM Completion Percentage		
Rep #	4 buffers	5 buffers	difference	4 buffers	5 buffers	difference
1	95.12	95.12	0.	92.31	92.31	0.
2	95.24	83.67	11.57	100.00	100.00	0.
3	92.68	95.83	-3.15	97.56	97.56	0.
4	97.22	97.30	-0.08	97.06	97.06	0.
5	93.48	96.43	-2.95	97.22	97.22	0.
6	100.00	93.88	6.12	91.67	91.67	0.
7	100.00	95.35	4.65	100.00	100.00	0.
8	87.18	91.89	-4.71	95.35	95.35	0.
9	100.00	94.29	5.71	97.73	100.00	-2.27
10	97.78	93.55	4.23	100.00	100.00	0.
11	94.74	95.56	-0.82	90.00	92.50	-2.50
12	100.00	100.00	0.	86.67	88.89	-2.22
13	96.88	88.24	8.64	85.42	87.50	-2.08
14	95.35	90.91	4.44	90.00	97.50	-7.50
15	95.45	86.79	8.66	100.00	100.00	0.
16	93.02	100.00	-6.98	97.56	97.56	0.
17	97.06	100.00	-2.94	97.22	100.00	-2.78
18	91.43	96.77	-5.34	91.11	95.56	-4.45
19	92.68	97.14	-4.46	95.65	97.83	-2.18
20	91.49	97.44	-5.95	97.14	100.00	-2.86

The Significance of the Observed Improvement

The variability of the differences is noticeably less using Syncsim than with Plainsim. The standard deviations of the differences in the table just given are 5.53 versus 2.00. This is a significant reduction in the standard deviation. Suppose that the primary purpose of the simulation runs is to measure the improvement in performance precisely by averaging over a (possibly) large number of replications. The standard error of the average from n replications is the standard deviation σ divided by $\sqrt{n}$. Therefore for Syncsim the estimated average difference is −1.442, with a standard error of .447. To obtain this same precision using Plainsim with a σ of 5.53, requires that n be increased from 20 to 153. This is over 7½ times as much simulating (the square of the ratio of the standard deviations). Synchronization has in this case effectively divided the overall running time by a factor of 7½.

Similar comparisons were made for each of six input parameters at an increased and decreased level. These were the probability of making a connection, number of buffers, slot length, interference event arrival rate, average message length, and triggering event arrival rate. The responses average time, completion ratio, and on-time ratio were used (see table). In all thirty-six cases the standard deviations of the differences were less using Syncsim. The table also gives in the "Benefit" columns the square of the ratio, which measures the increase in sample size required for Plainsim to achieve the same precision as given by Syncsim.

A further demonstration was made by employing a 36-run experiment design using all six of the key input parameters with the same three responses. A 28-term model was fit to the results. The residual standard deviations were found to be less using Syncsim. Moreover, the fitted models were more parsimonious and easier to interpret.

A key element of the approach is the use of two different random number generators, one to supply the starting seeds for the other. A test was developed adapted from Rantest7. At least for this example, it appears to be more critical that the generator used to supply the seeds be of high quality.

	Plainsim			Syncsim			Benefit		
Parameter	Time	Comp	SoS	Time	Comp	SoS	Time	Comp	SoS
P connect +	15.8	.071	.194	1.8	.011	.019	80	38	101
P connect −	14.2	.060	.172	2.0	.015	.040	53	16	18
Buffers +	12.0	.082	.177	4.1	.033	.044	8	6	17
Buffers −	17.1	.055	.199	1.6	.020	.032	119	8	38
Slot +	9.5	.047	.121	4.3	.020	.070	5	6	3
Slot −	11.3	.050	.138	3.6	.027	.072	10	3	4
Intrfer +	11.5	.046	.126	6.6	.030	.090	3	2	2
Intrfer −	13.4	.063	.176	4.8	.037	.079	8	3	5
Length +	14.3	.047	.180	3.4	.026	.078	18	3	5
Length −	15.3	.073	.210	4.4	.023	.075	12	10	8
Arrivals +	14.6	.065	.168	4.3	.019	.072	11	11	5
Arrivals −	10.6	.044	.128	5.2	.028	.074	4	2	3

A Recipe for Synchronizing Random Numbers

Synchronizing the random number draws is easy to do for any existing or new simulation, and it seems to be very beneficial in allowing comparisons to be made. It should be made a part of any simulation. The only exception is a simulation that always makes exactly the same random number draws by its design (which is effectively already synchronized). Here is how to do it.

1) Identify all the random number draws and associate each with an entity and a purpose. For example, in SPM entities might be nets, radios, hostile jammers, and operational facilities; purposes for a radio might include scheduler delay, probability of frame synchronization, and probability of collision.

2) Attempt to place random number calls in procedures before any branching, so that the same number of draws is made with each procedure call. A few wasted draws are of little concern.

3) Replace all random number draws by a call to a new routine RANDOM that gives as output the random number and accepts as input indices I for entity type, J for entity number, and K for purpose of call. Variants are possible; for Syncsim only J and K are used, with K values of 1 to 4 used for message arrivals, message recipient, message length, and probability of failure for radios, and K of 5 used for links.

4) Set up an array of seeds so that each IJK combination has its own seed. This may be done with a single triply dimensioned array or with a combination of arrays; the latter is particularly useful if there are disparate numbers of different entity types. Syncsim is dimensioned for a total of 49 radios and uses a 4×49 array for radios and a separate singly dimensioned array of length 1176 for all possible links between pairs of radios.

5) When called, routine RANDOM selects the appropriate seed array and seed, then uses it in a call to the actual generator. The new seed produced by the generator is returned to the seed array and the real number X returned to the caller.

6) There remains the task of initially filling the seed arrays. This may be done conveniently by means of another new routine RANSET, which is called at the beginning of the simulation run and again at the beginning of each new replication if more than one sample is included in the case. This routine uses a single seed as a user supplied input. It then uses a second random number generator, independent of that used in step 5, to fill in the seed arrays.

This procedure should be easy to implement for any simulation and give results that are easy to use and interpret. An ingenious alternative approach to step 4 that was developed by Jeff Niemuth is given in Appendix D. It uses a singly dimensioned array of seeds and an auxiliary doubly dimensioned array to accommodate differing numbers of entities of various types.

6. EXPERIMENT DESIGN FOR SIMULATION STUDIES

Motivation for Experiment Design

A simulation model usually has many input variables, and there generally comes a time when the user wants to explore what happens as these are changed. Assume that a reference set of input parameter values has been established, perhaps by modeling a baseline concept. The next step is often to run sensitivity analyses, in which one variable at a time is changed from its nominal value. This step is useful in establishing which of the many variables are the most influential, and in screening out some variables that are of no further interest, either because they have no effect on system performance or their effect is undesirable and the baseline value is the only suitable value.

A next logical step is to run further sensitivities, except that each variable that is still of interest is changed from nominal in the opposite direction than in the first set of sensitivity runs. Of course this only makes sense for variables that have a natural ordering and for which both a higher and lower value is meaningful for the simulation.

The question then remains as to what happens as several or many variables are changed from their nominal values. If only two or three variables are still of interest, then it makes sense to run all further combinations of levels that have not as yet been run. In this framework, we are dealing with three levels for each variable: nominal, high, and low. With two variables there are nine combinations, of which the five with at least one variable at its nominal value have already been run. The picture is completed by running the four remaining cases in which both variables are at either high or low values. With three variables, there are 27 combinations of which 7 have been run. Running the remaining 20 is a lot of work, but is not out of the question. With 4 or more variables, we need to look for an alternate approach.

There are several features of experiments with simulation models that do not jibe with the standard theory and practice of experiment design, which has been developed for experiments in the physical world. A key one is the choice of a reasonably sized set of input variable combinations to use for running simulation cases. For example, there are few suitable published designs for moderate numbers of variables all of which appear at three levels. Also, the usual selection criteria for finding designs depend on the error structure and on the terms appearing in the models.

Two areas within the general field of statistical experiment design are particularly relevant. Factorial design deals with the situation in which the independent variables are restricted to discrete levels, usually two or three. The purpose of factorial design is to obtain efficient estimation of parameters that are assumed to describe the response of the system at any combination of levels. The area of response surface design deals with a response that is an unknown function of several continuous variables. The value of the response may be estimated by running experiments at any combination of values within some region of interest. The purpose is often to find combinations of the independent variables that optimize the response.

Models to be Used

We will assume that the simulation model being used has some random components implemented by means of a pseudo-random number generator. For any combination of values of the input variables, the simulation gives a response that is subject to error because of the randomness. Presumably we could eliminate the random error by making a very large number of iterations, say as many as the cycle length of the random number generator, and averaging. In practice we will make a suitable number of iterations to drive the random error of the average down to a small value.

We will restrict attention to the situation in which the input variables are all numerical and are restricted to just three values. The full design obtained by running an experiment at every possible combination of levels is called the full factorial. The response for any set of input variables can be expressed in terms of a model that contains a grand mean, main effects, and interactions. The grand mean is the average response for all points of the full factorial. The main effects of any single input variable are the linear and quadratic components of the responses at the three levels of the input variable, averaged over all combinations of levels of all the remaining variables. The two-variable interactions are components of the responses at the values of two of the variables averaged over all combinations of the remaining variables. The two-variable interactions may be resolved into four components: linear-by-linear, two linear-by-quadratic, and quadratic-by-quadratic. Higher order interactions involving more than two variables may be defined analogously.

We will adopt as a working hypothesis that the response may be approximated adequately by just a subset of the terms. Specifically, we will assume that the subset contains just the grand mean, linear and quadratic main effects for each input variable, and linear-by-linear interactions between each pair of input variables. If this is the case, then it should be possible to calculate values for the operative terms from just a subset of the full factorial. Intuitively we expect that some subsets would be more suitable than others.

The response at any combination of input variables can be expressed as a linear combination of the unknown effect and interaction parameters plus error. In vector and matrix notation, let $\mathbf{Y}$ be a column vector of N responses from the experiment. Let β be the vector of k unknown parameters and let $\mathbf{e}$ be the N-component vector of errors. Then

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{e},$$

where the coefficient matrix $\mathbf{X}$, called the design matrix, is used to express the dependence of the responses on the parameters. In the usual case in which k is less than N and the matrix $\mathbf{X'X}$ is nonsingular, the least-squares estimate of β is given by

$$\beta = (\mathbf{X'X})^{-1}\,\mathbf{X'Y}.$$

If the components of $\mathbf{e}$ are independent and all have the same variance σ^2, then the covariance matrix of β is given by $(\mathbf{X'X})^{-1}\,\sigma^2$.

Designs for Simulation Studies

We have envisioned a simulation study involving a baseline case, followed by sensitivity runs each with one variable increased and then decreased, followed by more runs with several variables changed jointly. The experiment design for this sequence of cases would then have several features:

- It contains as a subset the points of the sensitivity studies
- It allows estimation of all main effects and all linear-by-linear interaction terms
- It has a moderate number of factors.

The experiment design literature does not seem to offer suitable designs sharing these special features. Therefore designs were developed for 3 to 10 factors, and are given in Appendix E. A summary of the designs (including the two-factor full factorial) appears below.

# factors	2	3	4	5	6	7	8	9	10
# parameters	6	10	15	21	28	36	45	55	66
# runs	9	15	21	27	35	44	54	65	77
Efficiency %	100	96	82	81	70	59	53	46	41

The designs were obtained and evaluated using an existing Experiment Design Evaluator written by the author. This interactive routine evaluates a given design, then gives the user an evaluation of which points might best be added to the design or deleted from the design. Iterative use of this exchange algorithm can often lead to improved designs. The criterion for improvement is related to the average statistical variation of a fitted response, where the average is taken over all points of the full factorial. This can be expressed in terms of the inverse of the cross-product matrix $\mathbf{X'X}$ as defined above. The criterion can also be considered in terms of how a subset design compares with the full factorial. The latter is fully efficient, but at the expense of being an extremely large design if the number of factors is not small. The efficiencies in this sense for the subset designs are included in the table. They are quite high considering the small sizes of the subset designs relative to the full factorial.

Application of Experiment Design to a Simulation

The simulation chosen for this experiment models an assistance telephone line for users of a widely disseminated computer software product. Customers call in with questions or problems for which they need expert assistance. Up to 5 service representatives are used to field their inquiries, but on any given day, up to two servers may be absent. The probability is .15 that exactly one server will be out, and .05 that exactly two will be out.

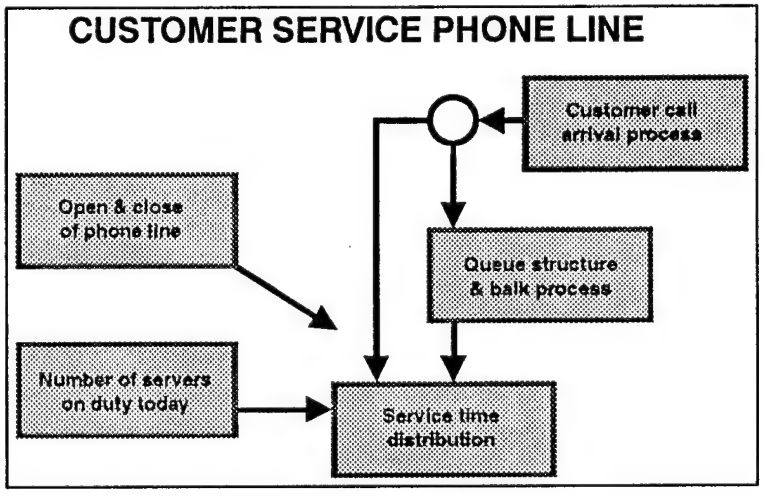
If all servers present are busy when a call arrives, the caller is put on hold. At this time a caller may decide that the wait isn't worth it and hang up. This *balk* probability depends only on the length of the queue. The balk probability is zero for a queue size of from 1 to 5, then increases linearly to be 1 at 10. Once a caller has decided to wait, he or she will continue to hold until served.

The phone line is configured with 10 hold positions. If a call arrives when these ten slots are already taken, it receives a busy signal and the caller must try again later. The line opens in the morning at 8, and runs all day until 5 in the afternoon. Calls arriving before 8 are put in the hold queue. Any calls that are on hold at 5 are retained and will eventually be answered by a server.

The arrival of calls will vary throughout the day. The arrivals are modeled as having independent exponential interarrival times with mean values as a function of time of day: 2.79 between 9 and 11, 2.79 between 1:20 and 4:20, and 6.44 at other times between 7:50 and 5. Service times will be independent of each other, of the time of day, and of how many callers are waiting. They will probably have a long-tailed distribution; this is assumed to be Erlang with parameter 2 and with mean service time of 9.5 minutes.

There are four responses of the system that are of interest for each day simulated. These are: 1) The number of customers served, which is a function of the arrival distribution and of the number of balks. 2) Maximum queue length that develops. 3) Utilization factor for the servers, expressed as the percentage of the time from 8 to 5 that the servers present are on the phone. 4) Average wait for service, expressed as total waiting time divided by the number of customers served (whether they actually had to wait or not).

This example is based on a slightly simpler example given by Bratley [1983]. Their context is arrival of customers at a bank to be served by tellers. Fortran code, again adapted from that provided by Bratley et al, was used to model the service phone line system.



The Experiment

The experiment involved seven input variables whose levels are in the table below. The design consisted of 44 points out of the $3^7 = 2187$ of the full factorial (2%). Simulations were run for 100 days each. The random number strings were synchronized so that the same random draws were made on each day no matter what the values of the input variables. The four responses were averaged over the 100 days.

Factor	Low Level	Nominal	High Level
A. Queue length without balk	3	5	7
B. Prob of absences 1, or 2	.10 .00	.15 .05	.20 .10
C. Opening time	7:40	8:00	8:20
D. Number of servers	4	5	6
E. Mean service time	8.5	9.5	10.5
F. Closing time	4:40	5:00	5:20
G. Mean interarrival times	6.12 2.65	6.44 2.79	6.76 2.93

The design points and values of the four responses are given in Appendix F, as are the estimated values of the parameters. Almost all of the linear main effects are appreciable, as is the quadratic effect of the number of servers. Ten of the 21 interaction terms also have appreciable influence on the responses. The fitted responses using the full 36-term model approximate the original observations quite well.

An example of one of the models resulting from this analysis, for total customers, is given here. All terms that are smaller than .15 (rounding to .0 or .1) have been omitted.

Effects:

Mean	144.9
A	1.1
B	−1.1
D	2.6
D quadratic	−.6
E	−1.3
F	3.2
G	−6.8

Interactions:

AB	.3
AD	−.8
AE	.5
AF	−.5
BD	1.1
BE	−.5
BG	.3
DE	1.2
DG	−.6
EG	.4

A Confirmatory Experiment

Point	Predicted responses				Observed responses			
0002111	147.03	3.12	42.18	0.19	146.84	3.07	42.02	0.27
0021111	145.72	5.52	53.92	1.27	145.55	5.49	54.02	1.53
0100010	148.98	5.75	61.59	2.92	149.12	5.79	61.89	2.91
0101000	150.45	4.25	50.73	0.78	150.28	4.33	50.72	1.00
0101010	153.57	4.18	49.94	0.62	153.50	4.33	49.95	0.98
0112100	150.58	4.12	48.49	0.49	151.03	4.00	48.61	0.67
0121111	144.40	5.67	56.55	1.86	144.66	5.57	56.66	1.97
0221011	145.19	5.69	53.22	1.73	145.04	5.52	52.91	1.79
0222122	142.72	5.43	45.88	1.33	141.88	5.09	45.44	1.09
1002101	144.51	3.44	43.03	0.27	143.85	3.24	42.72	0.31
1002122	142.58	2.74	39.39	-0.15	142.88	1.64	33.78	0.07
1011220	156.64	5.87	59.62	1.97	157.07	6.11	59.75	2.13
1012110	155.21	4.35	46.15	0.42	155.08	3.35	39.35	0.25
1021020	158.75	5.92	50.42	0.80	157.81	6.08	50.44	1.40
1122121	150.08	5.84	46.75	1.13	150.21	5.80	46.55	1.16
1212112	140.17	4.05	44.98	0.49	132.53	9.88	82.66	12.01
2020112	138.50	7.49	64.60	4.78	141.32	7.12	61.18	4.00
2120022	141.60	7.38	61.06	4.84	143.58	9.70	81.88	10.13
2120200	144.36	9.43	81.65	8.80	142.36	6.60	56.06	2.72
2221122	142.67	6.99	56.39	3.64	139.84	3.84	44.80	0.63

That the fitted model works well for the original design points is not surprising. The real question is whether the model works just as well for the 98% of the full factorial that is not part of the experiment. A test protocol was set up and followed exactly. Twenty points of the full factorial that were not contained in the original experiment were selected by using a published table of random numbers. Predictions were made using the model and simulations were run to get actual values of the four responses. The results are in the table above.

A meaningful comparison of the two sets of results may be obtained from the residual error of the responses from the first experiment, each having 8 degrees of freedom in the statistical sense, with the root mean square differences of predicted and observed from the confirmatory experiment, with 20 degrees of freedom. These are given in the table below.

Response	Customers	Max Queue	Utilization	Ave Wait
Residual (original experiment)	.545	.239	.256	.192
Prediction – Actual	.457	.200	.214	.494

The conclusion is that the experiment design gave very good results for the first three responses and pretty good results for the fourth. The reason the fourth response is not better is that its variability increases with its value. The logarithm of waiting time could be used instead to achieve more stable results.

A Scenario for Implementation of Experiment Design

The use of experiment design as illustrated appears to enhance the use of simulation models. In particular, it gives the analyst a good understanding of the behavior of the simulation model over a great number of combinations of inputs without actually running all the cases. We will explore briefly how it might be implemented in a semi-automated fashion.

Suppose that runs of a simulation model are set up and initiated by means of an interactive interface. The user may be presented with screens that request values for any key control parameters, and menus of sets of input parameters by means of which the user can establish default values, or change values for a particular run. Such a system could be extended to treat the interface with the experiment design. The user would be asked to specify which input variables are to be studied in the experiment and the levels for each. Output variables of primary interest for interpreting the results would also be specified. The user might also be given options as to how the results of the experiment are to be presented, such as raw estimates of effects and interactions, fitted value tables, plots, or estimated maxima.

The system would then take over and set up a series of simulation runs according to an experiment design established internally. The random number generation scheme for the basic simulation model would have been set up so that all random number draws are synchronized for each Monte Carlo replication of the model. In the phone help line simulation different strings are used for determining the number of servers absent, customer arrivals, queue balks, and service times. Random number synchronization is required for making meaningful comparisons between runs of the designed experiment.

Because many of the input parameter values are the same from one point to another in the experiment design, many of the same computations are done repetitively from one case to another. It should be possible to structure the simulation to take advantage of this fact. If an input variable is used only in a function subroutine, for example, then the subroutine could be modified to store its computed output value for each input used. At each call it would first check to see if it had been already done the requested computation, and if so just feed back the stored value. Restriction of the design to the factorial structure may mean that very few different values are actually used. It should be noted that use of an object-oriented approach to the simulation development, or at least a modular structure, would facilitate this saving.

7. OTHER GENERAL APPROACHES FOR INCREASED EFFECTIVENESS

The original thrust of the current effort was to examine approaches that might be generally applicable across a wide variety of simulations. It soon became apparent, however, that the techniques under examination could only be made effective by introducing approximations or decreases in fidelity. The work was therefore abandoned in favor of more promising efforts. Some of the techniques considered might still be useful for other types of simulations.

Scaling

The idea of scaling is used throughout scientific investigation. It is feasible to observe a small number of entities and their interactions by a single scientific investigator. The scientist then tries to generalize to systems consisting of larger numbers of entities. If interactions are few, then the scaling approach works: a dairy with 10 times as many cows will produce 10 times as much milk. Scaling is hopeless for some applications because of the complexity of the interactions, the two-body gravitation theory being an example.

Scaling was tried with the phone-line model described on page 36. The size of the model was increased by scaling up from 3 to 30 customer service representatives, and by increasing the number of customer arrivals correspondingly, in this case by dividing the interarrival rate by 10. Not surprisingly, the number of customers served per day did scale well. The average waiting time did not. In the larger model a server was much more likely to become available quickly, so that the waiting times were shorter and their distribution showed less spread. No general conclusions were reached, except that scaling is a complex issue.

Inverse Validation

Simulation validation methodology is used to determine how well a simulation matches reality. Suppose we have an existing simulation that has been validated but that we would like to speed up. Then we might use the results of the validation process to determine what parts of the model might be degraded without significant sacrifice of fidelity.

The idea here is to use a decomposition of the model into submodels for which data are available from both real system and the simulation. In such a situation, the validation of the overall model might be based on metrics that were weighted sums of metrics for submodels. As a system is developed, test data are often obtained for components and subsystems long before it is possible to make any tests on the whole system. The system model might then be speeded up by substituting faster running submodels for those that are slow running but of greater complexity than needed for the overall simulation. The inverse validation principle would structure the choice of where to compromise fidelity.

When an engineering simulation model is developed in parallel with the system development, it is common to have submodels of varying degrees of fidelity. Rather than decreasing the fidelity of those that are high, the spirit of engineering simulation is to leave them at high fidelity. For other types of simulations, such as system effectiveness models, this approach might be useful.

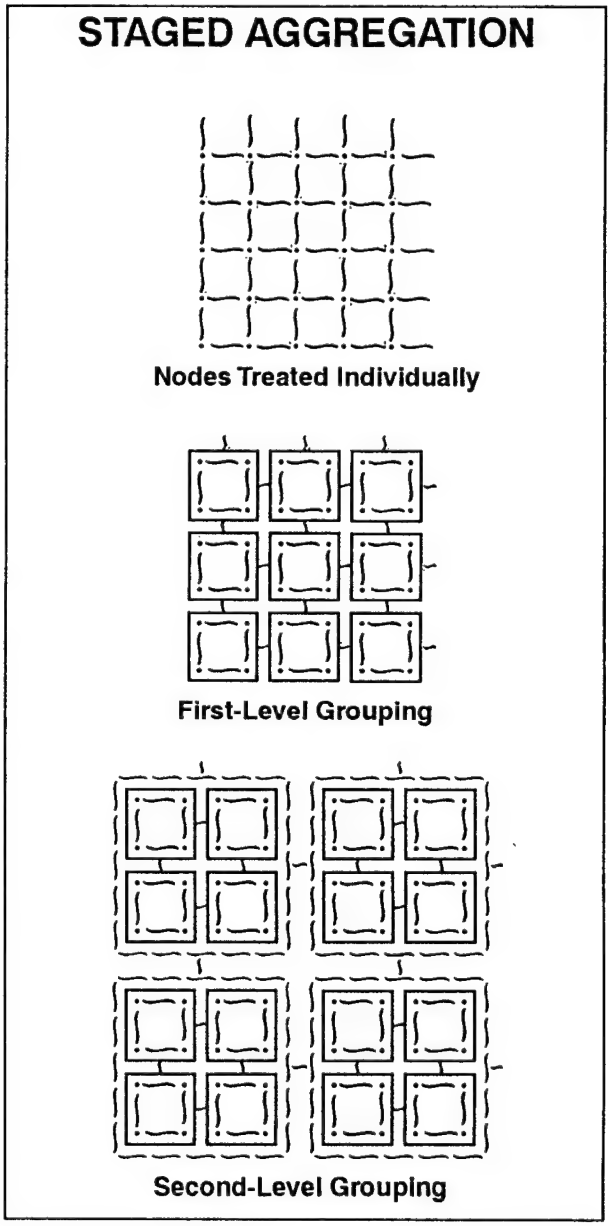
Staged Aggregation

A communication system simulation is characterized by having a large number of entities of the same or similar types that are all linked together. A simple example is a system consisting of many terminals that send messages to each other. If the system is to be simulated, the straightforward approach is to simulate each terminal and the individual messages between them. As the system grows in size, the size and corresponding run times of a simulation grows in a combinatorial fashion.

Aggregation may be applied to combine entities into groups of small size. The result might now be considered a higher-order building block. Then further growth is modeled by linking these next order building blocks, and the process repeated for a few stages. Thus, for example, rather than modeling a system of 1000 terminals, the system is represented as 10 supersets of 10 groups of 10 terminals.

Continuing this reasoning, the system consists of a small subsystem, operating in parallel and interacting with many replicas of itself. The corresponding modeling approach is to represent the subsystem in terms of its own internal transactions, together with interactions with its replicas. Thus, for the networked terminal model the subsystem model would explicitly account for messages sent and received within itself, the messages sent to other replica subsystems, and the messages from other replica subsystems.

As part of this contract, some work was done on a network model to test out this idea. One node in a network initiates a message that it sends to its neighbors, who relay the message on. Fortran simulation models were written to compare levels of aggregation up to the third level (figure), but the complexity of the aggregated models seemed to increase faster than the potential savings and most results were negative. The work done is summarized in Appendix G.



8. RECOMMENDATIONS AND IMPLEMENTATION PLAN

Random Number Synchronization

It is recommended that a random number synchronization scheme be implemented for CECOM simulation models, including SPM in particular. This is easy to do and can be done by those who routinely maintain the models following the procedure given on page 32.

With a synchronization scheme implemented as indicated, one seed controls the generator that produces the seeds for the generator actually used during the simulations. It is recommended that that seed be a user controlled input parameter. If a specific seed is to be used in the current SPM implementation, code must be recompiled.

If synchronization is *not* implemented, then separate runs should be made with different seeds. This is to avoid the situation in which runs are essentially the same for awhile, then diverge after random number synchronization is lost. Meaningless comparisons might result as discussed on page 24. The technique used in SPM of setting the seed by the low-order bits of the system clock is effective and should be retained.

If synchronization is *not* implemented, then the option of using statistically generated input message generation should not be used for cases to be compared to each other. The input should be generated offline, using the statistical technique if desired. The resulting message set should then be used as a scripted input for the actual simulation cases. This is because of the strong dependence of a communications simulation on the input traffic. In fact, for many such simulations most of the benefit of the random number synchronization technique might be achieved by using scripted inputs.

Other Recommendations

An automatic timing system should be introduced into simulations. This can be done by calling the system clock when each major procedure is initiated, again at the end, and cumulating the time increments so obtained. The total time in each procedure should then be printed as part of the simulation output. Timing data can be refined for those procedures that use the most time. The resulting data will be useful in directing future efforts at optimizing simulation run time.

Although the current random number generator in SPM is judged to be adequate, it is not the best generator available. Its replacement with a proven generator would add to credibility. The generator called GEN_H or GEN_A in Appendix B might be good choices here, subject to further testing. The best generator studied, GEN_K, is more difficult to initialize because it stores intermediate values internally, and incidentally requires more computer time.

An automated system to set up simulation cases following a statistical experiment design should be pursued. The usefulness of this step is contingent on having a random number synchronization scheme in place. Details need to be worked out including what the experiment designs should be and what the user interface should look like. A better generator is strongly suggested as an adjunct to an automated experiment design capability.

Implementation Plan

It is frequently suggested that a follow-on contract would be the best vehicle for implementing recommendations arrived at in a study. In this case, however, most of the implementation can be done by those responsible for the model and those performing routine model maintenance. It is assumed that SPM is the intended target for implementation, but other models could be used just as well.

First, two interim operating procedures should be established applicable to production runs with the SPM. A production run is any that will contribute to analyses or that will be used as the basis for decisions. Different cases should use different random number seeds. Generation of the input message scenario should be done offline and the result used as a scripted input that is archived. Even if no comparisons are contemplated with a particular case, the script is available should a later comparison need to be made.

Second, synchronization of the random number draws should be implemented. The first step is to identify all random number draws and associate with each an entity type, entity number, and purpose. In performing this association, it is important to separate cases in which different numbers of draws might be made depending on different input parameters or on different conditions. If the same number of draws will always be made, then purposes may be lumped together.

The actual coding phase follows the procedure outlined on page 32. This involves a new random number provider that accepts indices, an array of seeds, an initialization procedure, and two independent random number generators. All calls to the existing random number generator (in any form) are replaced by calls to the new random number provider. For SPM this will involve replacing the calls to Uniform, Expon, Texpon, Normal, and Tnormal with new equivalents. Random number draws should be moved back before IF structures so that the same number of draws will always be made.

Testing would include tests that the procedure is implemented properly and also regression testing in which inputs are set up to match old cases that have been studied in the past. Tests would also be made with parameters changed to verify that close comparisons are now possible.

Once synchronization is in place, the single seed that drives the random number generator that fills the seed array for the other generator should be made part of the user input, and no longer generated via the system clock.

As a separate effort, timing instrumentation should be added to the code so that in the future efforts to increase running speed can be concentrated on the routines that use the most time.

A facility for automating the running of statistical experiment designs appears to offer potential benefits. Further study of implementation details, particularly the analyst interface, and the benefits using it are required.

APPENDIX A - VALIDATION METRIC ROUTINES

Validation of a simulation model is often based on a comparison between data from a test of the real system and corresponding data from the simulation. Metrics that express the degree of agreement have been developed and were implemented in the form of computer routines for use in validating SPM. Details of the theory, routines, and testing are given in this appendix.

A Metric for Binomial Data

Statistical procedures are available for testing hypotheses about whether two binomial proportions are equal, or for establishing confidence intervals on single binomial proportions. Methods do not seem to be readily available, however, for determining the confidence with which two proportions lie within a specified interval. A procedure was developed for this purpose.

If the unknown probability of success on any one trial is viewed as having a probability distribution which is uniform between zero and one before data are obtained, then after observing K successes in N trials, the parameter p has a beta distribution. Specifically, the posterior probability that p is less than any value x between 0 and 1 is given by

$$\Pr\{p < x\} = (N + 1) \binom{N}{K} \int_0^x t^K (1-t)^{N-K} \, dt.$$

The figures given on page 15 show the shape of the probability density for representative values of N and K. In particular, for small N the density varies slowly and is significantly above zero for much of its width. For large N, however, the density is extremely peaked, but has negligible values except near the peak.

Using this approach, the probability that two binomial proportions p_1 and p_2 are within $\pm d$ of each other can be calculated from a convolution of two beta distributions. This can be expressed symbolically as $\Pr\{p_1 = y\} \times \Pr\{y-d < p_2 < y+d\}$, integrated over all possible values of y. Evaluation of this expression involves an outer integration from 0 to 1 of the density of p_1, obtained from the above beta expression using N_1 and K_1, and an inner integration between $y-d$ and $y+d$ of the beta density using N_2 and K_2.

The numerical evaluation of this expression over wide ranges of the five parameters is challenging. The integrals are replaced by summations with finite step sizes used for the infinitesimals dy and dt. The step sizes used must be small enough to give satisfactory accuracy, but not so small as to give unacceptably large computation times. If N is small, a fairly large integration step size will give accurate results. If N is large, however, the density is very peaked and a small step must be used. On the other hand, the density is negligible over part of the range. In any case, the densities are unimodal.

These features have been taken advantage of in several ways:

• If the density in the outer integral is so small that the resulting term will be negligible, then the inner integration is skipped.

• If the contribution of the current term of the outer integrations is a small fraction ε_1 of the total already achieved, then the process is terminated.

• If the contribution of the current term to the outer integration is a small fraction ε_2 (larger than ε_1) of the total, then the integration step size for the inner integral is increased by multiplying by 2, up to a limit of 2^6.

• The direction of integration is controlled so that the peak of the density is treated first and the long tail last.

A second problem is the evaluation of the binomial coefficients for N things taken K at a time, which is equal to N! / K! (N–K)!. If N is large then this expression will cause computational overflow except when K is close to N or 0. The approach used is to make a preliminary check to see if the expression will be large, and if so, then use an alternative computation based on the Gaussian approximation to the binomial. The Gaussian approximation does not involve explicit integration, and is very fast.

A Fortran routine using these features, called BETALIM, was sent to John Wray of AMSAA for his evaluation. The basic integration step size was set to .0001. On a 486 computer (at 33MHz) it took 44 seconds to solve 10, 9, 10, 9, .01, but only a couple of seconds to solve 200, 200, 200, 200, .01. The accuracy of the first seems to be about .0002, but of the second only .01. If the step size is decreased to .00001, the accuracy is better but the times are increased by a factor of about 60.

A further problem found by Wray was that the program stopped with an exponentiation error when K was equal to N. This is probably due to a compiler difference in the treatment of 0^0 when evaluating the density.

A second version, called BETALIM2, was created and also sent for evaluation. It treats the 0^0 case specially, uses a variable step size depending on the larger value of N, and uses the more accurate Simpson's rule for the inner integration rather than the trapezoidal rule. The technical control parameters were tuned to attempt to give 3-place accuracy to the result over wide ranges of the inputs. A test case was constructed with 13 sets of inputs, based on a sample data set provided by Wray. N_1 and N_2 were equal, ranging from 116 to 2190, K_1 and K_2 were also equal, ranging from N_1 down to $N_1 - 2$, and the indifference interval d was .05. This test case took 22 minutes and 34 seconds on the 486 computer. The indifference interval of .05 is large (all the computed probability levels are essentially one), and smaller values would take less time.

Further examination of the large-sample approximations used is in order. The test made is based on an approximation to the Stirling approximation to the factorials in the binomial coefficient; specifically, the quantity TEST given by

$$TEST = N \ln N - K \ln K - (N-K) \ln (N-K)$$

is compared with a threshold value, currently set to 50. For values of N less than 73 this results in the exact formulas being used. For larger values of N the region for which the approximations kick in is given by central values of K:

N = 75	K between 30 and 45
N = 80	K between 26 and 54
N = 100	K between 20 and 80
N = 200	K between 14 and 186
N = 10,000	K between 6 and 9994.

It should be noted that the region is a function of both N and K, and not just applied when N is large.

A further refinement, BETALIM3, incorporated separate calculation of step sizes for the two integrations, and more accurate large-sample approximations. Unfortunately, it proved to have some numerical problems, so it was withdrawn from consideration in favor of BETALIM2.

An accuracy statement is given by Mood [1950] for the Gaussian approximation to that of the binomial proportion. The statement is made that the error is less than $.15/\sqrt{N\,p\,q}$ (where p is the true underlying probability of success and q is 1 – p), provided that N p q is greater than 25. Although this sounds accurate, it really isn't. The values of N p q at the border between where the exact formula or approximation is used range from around 18 at N = 75 down to 11½ for an N of 300, on down to 6 for N = 10,000. A check of values chosen along the boundary in such a way that the first sample would use the integration and the second the approximation was made using both BETALIM2 and BETALIM3. The results are in the following table, rounded to 4 places from the original output.

N_1	K_1	N_2	K_2	d	BETALIM2	BETALIM3	Difference
73	49	73	41	.02	.0806	.0803	.0003
74	49	74	73	.02	.1208	.1214	–.0006
75	49	75	46	.02	.1783	.1808	–.0026
80	59	80	54	.02	.1515	.1543	–.0029
130	119	130	113	.02	.2020	.2092	–.0073
200	191	200	186	.02	.3768	.3941	–.0172
300	295	300	288	.02	.3997	.4170	–.0173

This study is too limited to be definitive, but gives an indication of possible accuracy.

A Metric for Delay Time Data

Because the message delay time data are continuously variable rather than restricted to two values, a different processing procedure was required. Several different formulations were tried for judging whether a particular arrangement of X's and Y's is more extreme than the actual one observed.

The original approach developed under an earlier contract was intended for the case in which only a few observations were available from the real system. The procedure is based on the assumption that the two populations have the same distribution except for a location parameter. First we assume that there are two observations available from the real system and any practical number from the simulation.

Let $X_1, X_2, \ldots, X_m$ be an ordered sample of results for a single response from the simulation, where m is large; let Y_1, Y_2 be an ordered sample of size 2 from the real system. Assume that the random variable $Z = Y + \theta$ has the same distribution as the X's. Then all possible permutations of the $m + 2$ variables consisting of the X's and Z's are equally likely, each with probability $1 / B(m+2,2)$, where $B(n,r)$ is the notation for the binomial coefficient of n things taken r at a time. By counting arrangements we find

$$\text{Prob}\{Z_1 < X_i\} = (m-i+2) / B(m+2,2), \quad \text{and} \quad \text{Prob}\{Z_2 < X_i\} = i / B(m+2,2).$$

These can be used to make confidence statements about θ of the form $\text{Prob}\{\theta < X_i - Y_j\}$.

If the indifference zone on θ is $\pm d$, then the set of differences is searched to find values of i and j for which $-d \le X_i - Y_j$, and for which $X_i - Y_j \le +d$. The probabilities are evaluated from the above formulas, and the difference taken to bound the confidence with which θ lies between $-d$ and $+d$. The choices are made to maximize this difference.

This original formulation had the disadvantage that if the roles of X and Y are reversed, a different answer is obtained. A second problem was that the answer also changed if all data were subtracted from 100, but an adjustment was formulated that corrected this problem.

An alternative formulation considers counts of all possible arrangements for which all the Y ranks are less than or equal to those observed. This formulation does have symmetry if the roles of X and Y are interchanged, but presents a more challenging counting problem. An innovative recursive approach was formulated, but is an alternating summation of terms of alternating sign. The numerical errors inherent in this approach prevent correct computation for sample sizes n greater than about 15 or 20.

The approach finally selected uses the Mann-Whitney U statistic, which is a count of the number of instances in which a member of the second sample is less than a member of the first. The value of U can range from 0 if all the observations in the second sample are greater than any in the first, to $N_1 \times N_2$ if all are less than any in the first. If two samples are very different, then U will have a value close to one of these extremes. If they are the same, then

U will probably have a central value. If a sample of N_1 X's is really from the same population as a second sample of N_2 Y's, then if all the $N_1 + N_2$ observations are sorted, then any particular pattern of X's and Y's is equally likely to occur. The probability distribution of U in this case can be computed from a recursion relationship giving the number of possible arrangements of N_1 X values and N_2 Y values that give the same value for U. Assume for definiteness that $N_2 \leq N_1$. If $N_2 = 1$, then there is just one arrangement of the N_1 X values and 1 Y value with each of the possible values of U from 0 to N_1. Let the notation $MW(u; N_1, N_2)$ be the number of arrangements that give the value u for the statistic U. Then the recursion is

$$MW(u; N_1, N_2) = MW(u; N_1\text{-}1, N_2) + MW(u\text{-}N_1; N_1, N_2\text{-}1),$$

where $MW(u; N_1, N_2)$ is interpreted to be 0 if $u < 0$ or if N_1 or N_2 is less than or equal to 0.

For large values of N_1 and N_2 a Gaussian approximation is available. This is based on the asymptotic distribution, but is considered to be "reasonably" accurate for equal sample sizes as small as 6.

The derivation thus far assumes that the measured values are from a continuous distribution, so that ties do not occur. In practice, values are only recorded to some number of significant digits, and ties may occur. The statistic U may be modified so that each time a Y is less than an X, 2 points are scored, and if a Y is tied with an X, 1 point is scored. With this formulation, U may range from a minimum value which is again 0 to a maximum which is $2 N_1 N_2$.

To form a metric giving the confidence that two samples represent populations that are within an indifference $\pm$ d of each other in location parameter, the U statistic is computed twice using the second sample values with d added and subtracted. The values of U are compared with the percentage points of its distribution to obtain values that are differenced to form the final metric.

A Fortran routine called METRIC7 was developed implementing this procedure and sent to AMSAA for evaluation. Only the large-sample approximation was implemented for the initial delivery. Another version, METRIC8, was developed that improves the large-scale approximation slightly, but more importantly adds the exact computation for cases in which both sample sizes are less than 20. This routine works by building a table of the exact distribution, which is referenced for particular values of U. Attempts were made to develop a version that would use the recursion relationships to obtain distribution values as they are needed. If successful, this approach could have covered the cases when only one sample size is less than 20. The implementation involved making a procedure call for each term in the recursion, so that a deeper and deeper stack of calls was made as the evaluation proceeded. The attempts failed to achieve results in reasonable amounts of computer time except for small sample sizes, for which METRIC8 could be used. Therefore the approach was abandoned. The case in which one sample size is less than and one greater than 20 occurs rarely if at all for the EPLRS data, so METRIC8 was used for the data reduction for the VV&A effort.

APPENDIX B – RANDOM NUMBER GENERATOR TESTS

The Tests

A battery of tests for random number generators was constructed as follows:

• Rantest - Inspection. Prints out the first three uniform draws and corresponding seeds for the generator under test.

• Rantest2 - Cycle Length. Makes up to 30,000 draws and checks each successive seed value against a stored list of the last 20,000 seed values. Impractical to use.

• Rantest3 - Pair Uniformity. Draws 20,000 pairs and sorts them into a 64×64 grid, then makes chi-square test for uniformity.

• Rantest4 - Gaps. Tests distribution of 7000 gaps of length up to 70 successive draws, where the gap intervals are 0.0 to 0.1, .15 to .25, .45 to .55, .75 to .85, and .9 to 1.0.

• Rantest5 - Permutations. Draws 5000 sets of 6 numbers and classifies which of 720 permutations their ordering falls into.

• Rantest6 - Runs. Classifies 40,000 runs of increasing size of length up to 8; that is , if $X_1 < X_2 < X_3 > X_4$, then the run is of length 3.

• Rantest7 - Overlapping Triples. Sorts 30,000 numbers (in circular string) by which bin of an $8 \times 8 \times 8$ grid each successive triplet falls into (a Marsaglia 'stringent' test).

Fortran listings of tests 3 through 7 are given later in this Appendix.

The Generators

The tests were applied to a set of 14 random number generators, for which listings are also given later. These are:

• UNIF - Park & Miller generator as given by Kruger [1990].
• UNIF2 - Portable generator given by Bratley [1983].
• GEN_A - This is a portable generator written by Robert Guy (currently with Kaman Sciences Corporation in Colorado Springs) to emulate the generator RAN used in the Fortran library of the VAX 11/780 (called RANX by Guy).
• GEN_B - Full precision implementation of a linear congruential generator (LCG) with a = 3141592653 and c = 2718281829, given as Generator B by Knuth [1969, p 40] apparently as an example of an arbitrary choice of constants.
• GEN_C - According to Knuth this and GEN_D have been discussed in the literature, but they perform poorly because their multipliers are too small.
• GEN_D - The multiplier of 23 is much too small.

PARAMETERS FOR CONGRUENTIAL GENERATORS

Generator	a	c	m
UNIF	16807	0	$2^{31} - 1$
UNIF2	16807	0	$2^{31} - 1$
GEN_A	69069	1	2^{32}
GEN_B	3141592683	2718281829	2^{35}
GEN_C	129	1	2^{35}
GEN_D	23	0	$10^8 + 1$
GEN_F	262145	1	2^{35}
GEN_G	16807	0	2^{32}
GEN_H	630360016	0	$2^{31} - 1$
GEN_I	62605	0	2^{29}
GEN_J	69069	0	2^{32}
GEN_L	65539	0	2^{31}

- GEN_E - A Fibonacci generator without lag; that is, $X_{n+1} = X_n + X_{n-1}$. This formulation is known to give poor results.
- GEN_F - A generator with $a = 2^{18} + 1$, which can be shown to be unsatisfactory from number theory.
- GEN_G - A naive (incorrect) implementation with a = 16807 (same as UNIF and UNIF2).
- GEN_H - This multiplier is used in SIMSCRIPT II.5 and in DEC-20 FORTRAN according to Bratley [1983].
- GEN_I - Generator with a = 62605 used in the Berkeley Unix Pascal generator and found by Marsaglia [1985] to be "not so good" on a stringent test not used here.
- GEN_J - Uses a multiplier given by Marsaglia as a "failure bordering on the spectacular" for a stringent test.
- GEN_K - Marsaglia's combination generator using a multiplicative Fibonacci type and a difference lag-1 Fibonacci, combined by difference.
- GEN_L - An implementation using the multiplier 65539 used in the infamous IBM generator called RANDU.

Testing Procedure

Each generator was tested with the following set of 23 initial seeds:

Beginning integers: 0, 1, 2, 3, 4.
Choices the author has often used: 123456789, 1111111, 6999.

Powers of 2: 65536, 16777216, 1078741824.
Powers of 10: 10, 100
More choices sometimes used: 194305786, 1217344457, 314159276, 543219876, 9571916, 5868958.
Choices made by Bratley et al: 1234567890, 1933985544, 2050954260, 918807827.

The seed 1078741824 was used mistakenly; the 30th power of 2 is really 1073741824. The congruential generators with c = 0 do not work with the seed 0, so it was omitted for them.

The Rantests were not written with a general call, but must be edited to change the generator call to the specific one to be tested, then recompiled. A general facility using string manipulation might be desirable here.

Most of the tests took about a minute of computer time on a 486, with the range being 20 seconds to 2 minutes. The combined generator GEN_K usually took almost twice as long as the others. This is not surprising, since it effectively combines two separate generators.

Verification Testing on the Rantest Routines

Rantest3

For this test 300 pairs of numbers using generator GEN_K and seed 2 were sorted into a 10 x 10 grid. Categorization was hand checked. Computation of chi square was checked.

Rantest4

This test looked at 20 gaps of length up to 10 using generator GEN_A and seed 2. Only the .0 to .1 and the .9 to 1.0 gaps were checked (these were easiest to hand count). The counts agreed; the computation of chi square agreed; the P values seemed good with a rough table lookup using the 9 degrees of freedom (formula assumed > 30 df, but still looked like about 3 places accuracy).

Rantest5

A special run was made with 100 sets of 4 draws, using generator GEN_B and seed 2. With each draw was printed the permutation number, of the 24 possible. It was verified that numbers were assigned uniquely to permutations. These were as follows:

dabc cdba cadb cabd dcab bdac bcda bcad dacb bdca badc bacd
1 2 3 4 5 6 7 8 9 10 11 12

dbac cdab cbda cbad dcba adbc acdb acbd dbca adcb abdc abcd
13 14 15 16 17 18 19 20 21 22 23 24

The counts of these were hand checked. Computation of chi square was verified.

Rantest6

A special run was made with 100 runs of length up to 4, using seed 2 with GEN_L. The results were hand checked to compare with the run distribution obtained by the software. Runs were 49 of length 1, 24 of 2, 22 of 3, and 5 of 4 or more. The probabilities of runs of these lengths are 1/2, 2/6, 3/24, and 1/24. Chi square value is 10.02, which checks. Degrees of freedom = 3. x = 3.1654384, Z(x) = .0026613, P(x) = .999223, P value = .01840 by hand computation using the odd degree of freedom formula programmed and interpolation in table for P(x).

Random Number Generator Listings

```
      FUNCTION UNIF2(IX)
      INTEGER*4 IX, K1
C
C  PORTABLE RANDOM NUMBER GENERATOR FROM BRATLEY, FOX & SCHRAGE p319
C
C  USES IX = 16807 * IX MOD(2**31 -1)
C  PROBABLY REQUIRES DECLARATION INTEGER*4 IX, K1
C
C  INPUT: IX = INTEGER > 0, < 2147483647
C
C  OUTPUT: IX = NEW RANDOM INTEGER
C        UNIF = UNIFORM FRACTION BETWEEN 0 AND 1
C
      K1 = IX / 127773
      IX = 16807 * (IX - K1*127773) - K1 * 2836
      IF (IX .LT. 0) IX = IX + 2147483647
      UNIF2 = IX * 4.656612875E-10
      RETURN
      END
*************************************************************************
      REAL FUNCTION UNIF(ISEED)
C
C  PARK & MILLER GENERATOR.  FROM KRUGER: EFFICIENT FORTRAN PROGRAMMING
C
      INTEGER*4 KA, KQ, KR, M, ISEED
      INTEGER*4 KHI, KLO, TEST
      DATA KA /16807/, M /2147483647/, KQ /127773/, KR /2836/
C
      KHI = ISEED / KQ
      KLO = MOD(ISEED,KQ)
      TEST = KA*KLO - KR*KHI
      IF (TEST .GT. 0) THEN
        ISEED = TEST
      ELSE
        ISEED = TEST + M
      END IF
      UNIF = REAL(ISEED) / REAL (M)
      RETURN
      END
*************************************************************************
      FUNCTION GEN_A(ISEED)
C  WRITTEN BY BOB GUY AS RANX TO EMULATE THE VAX ROUTINE CALLED RAN
      REAL*8 RAND
      REAL*8 XSEED,YSEED
      INTEGER*4 ISEED
C       PARAMETER CNST = 69069.D0
       DATA CNST/ 69069.0D0 /
C
```

```
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = (XSEED*CNST) + 1.D0
      XSEED = YSEED -(ANINT(YSEED / 2.D0**32) * 2.D0**32)
      IF (XSEED .LT. 2.D0**32) THEN
        ISEED = INT(XSEED)
      ELSE
        ISEED = INT(XSEED - 2.D0**32)
      ENDIF
      XSEED = INT(XSEED / 256.D0)
      RAND = XSEED / (2.D0**24)
      IF (RAND .LT. 0.D0) RAND = 1.D0 + RAND - 5.9064645D-8
       GEN_A = RAND
C      RANX = RAND
C      write(3,'(f9.7)') ranx
      RETURN
      END
*******************************************************************************
      REAL*8 FUNCTION GEN_B(ISEED)
      INTEGER*4 ISEED
      REAL*8 A, C, XMOD, YSEED, XSEED
      DATA A / 3141592653.D0 /, C / 2718281829.D0 /
      XMOD = 2.D0 ** 35
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = DMOD(XSEED*A + C, XMOD)
      ISEED = INT(YSEED)
      GEN_B = YSEED / XMOD
      RETURN
      END
*******************************************************************************
      REAL*8 FUNCTION GEN_C(ISEED)
      INTEGER*4 ISEED
      REAL*8 A, C, XMOD, YSEED, XSEED
      DATA A / 129.D0 /, C / 1.D0 /
      XMOD = 2.D0 ** 35
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = DMOD(XSEED*A + C, XMOD)
      ISEED = INT(YSEED)
      GEN_C = YSEED / XMOD
      RETURN
      END
*******************************************************************************
      REAL*8 FUNCTION GEN_D(ISEED)
      INTEGER*4 ISEED
      REAL*8 A, C, XMOD, YSEED, XSEED
      DATA A / 23.D0 /, C / 0.D0 /
      XMOD = 10.D0**8 + 1.D0
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = DMOD(XSEED*A + C, XMOD)
      ISEED = INT(YSEED)
C      WRITE(3,99) XMOD, XSEED, YSEED
C  99 FORMAT('DEBUG XMOD, XSEED, YSEED',3F12.1)
      GEN_D = YSEED / XMOD
      RETURN
      END
*******************************************************************************
      REAL*8 FUNCTION GEN_E(ISEED)
C  FIBONACCI GENERATOR
      INTEGER*4 ISEED
      REAL*8 A, C, XMOD, YSEED, XSEED
      DATA A / 3141592653.D0 /, C / 2718281829.D0 /
      XMOD = 2.D0 ** 32
```

```
      XSEED = A
      A = ISEED
      IF (A .LT. 0.D0) A = A + 2.D0 ** 32
      YSEED = DMOD(XSEED + A, XMOD)
      ISEED = INT(YSEED)
      GEN_E = YSEED / XMOD
      RETURN
      END
*******************************************************************************
      REAL*8 FUNCTION GEN_F(ISEED)
      INTEGER*4 ISEED
      REAL*8 A, C, XMOD, YSEED, XSEED
      DATA A / 262145.D0 /, C / 1.D0 /
C  A IS 2**18 + 1
      XMOD = 2.D0 ** 35
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = DMOD(XSEED*A + C, XMOD)
      ISEED = INT(YSEED)
      GEN_F = YSEED / XMOD
      RETURN
      END
*******************************************************************************
      FUNCTION GEN_G(ISEED)
C
C  THIS AND NEXT ARE IMPLEMENTATIONS OF MULTIPLIERS MENTIONED BY
C  BRATLEY et al ON p 184.  THIS ONE IS USED IN APL, IMSL, AND SIMPL/I
C
      INTEGER*4 ISEED
      DATA A / 16807. /
      XMOD = 2147483647.D0
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = DMOD(XSEED*A, XMOD)
      ISEED = INT(YSEED)
      GEN_G = YSEED / XMOD
      RETURN
      END
*******************************************************************************
      FUNCTION GEN_H(ISEED)
C
C  THIS MULTIPLIER IS USED IN SIMSCRIPT II.5 AND IN DEC-20 FORTRAN
C
      INTEGER*4 ISEED
      REAL*8 A, XMOD, XSEED, YSEED
      DATA A / 630360016. /
      XMOD = 2147483647.D0
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = DMOD(XSEED*A, XMOD)
      ISEED = INT(YSEED)
      GEN_H = YSEED / XMOD
      RETURN
      END
*******************************************************************************
      FUNCTION GEN_I(ISEED)
C
C  MENTIONED ON p 7 OF MARSAGLIA. BERKELEY UNIX PASCAL. NOT SO GOOD
C
      REAL*8 A, XMOD, XSEED, YSEED
      INTEGER*4 ISEED
      DATA A / 62605. /
      XMOD = 2.D0 ** 29
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
```

```
      YSEED = DMOD(XSEED*A, XMOD)
      ISEED = INT(YSEED)
      GEN_I = YSEED / XMOD
      RETURN
      END
*****************************************************************************
      FUNCTION GEN_J(ISEED)
C
C  MENTIONED BY MARSAGLIA ON p 7. "SPECTACULAR FAILURE"
C
      REAL*8 A, XMOD, XSEED, YSEED
      INTEGER*4 ISEED
      DATA A / 69069. /
      XMOD = 2.D0 ** 32
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
      YSEED = DMOD(XSEED*A, XMOD)
      ISEED = INT(YSEED)
      GEN_J = YSEED / XMOD
      RETURN
      END
*****************************************************************************
      FUNCTION GEN_K(ISEED)
C
C  COMBINATION GENERATOR  COMBO  GIVEN BY G. MARSAGLIA p9 IN
C  COMPUTER SCIENCE AND STATISTICS: THE INTERFACE, 1985
C
      INTEGER*4 ISEED, IX0, IX1, IX2, JY0, JY1, JY2, JY3
      REAL*8 XMOD, YMOD, DIFF
C
C  CHOICES OF STARTING VALUES AT RANDOM.  IX ODD
C
      DATA IX1 /  1406829 /, IX2 /  7843281 /,
     &     JY2 / 15272794 /, JY3 / 11523568 /
C     DATA IX1 / 14728051 /, IX2 / 13225497 /,
C     &     JY2 / 15652424 /, JY3 / 17735477 /
C     DATA IX1 /  5525003 /, IX2 /  2481953 /,
C     &     JY2 / 18476168 /, JY3 /  7136408 /
C
      XMOD = 2.D0 ** 32
      YMOD = 2.D0 ** 30 - 35.
      JY1 = ISEED
      IF (JY1 .LE. 0) JY1 = JY1 + YMOD
      IX0 = MOD(IX2 * IX1, XMOD)
      JY0 = MOD(JY3 - JY1, YMOD)
      DIFF = IX0 - JY0
      IF (DIFF .LE. 0.) DIFF = DIFF + XMOD
      GEN_K = DMOD(DIFF, XMOD) / XMOD
C
C  MOVE STACKS DOWN
C
      JY3 = JY2
      JY2 = JY1
      ISEED = JY0
      IX2 = IX1
      IX1 = IX0
      RETURN
      END
*****************************************************************************
      FUNCTION GEN_L(ISEED)
      INTEGER*4 ISEED
      REAL*8 CNST, XSEED, YSEED
      DATA CNST / 65539.0D0 /
      XSEED = ISEED
      IF (XSEED .LT. 0.D0) XSEED = XSEED + 2.D0 ** 32
```

```
      YSEED = DMOD(XSEED*CNST,2.D0 ** 31)
      ISEED = INT(YSEED)
      GEN_L = YSEED / (2.D0**31)
      RETURN
      END
```

Random Number Test Program Listings

```
      PROGRAM RANTEST3
C
C  SORTS  LIM  PAIRS OF UNIFORM NUMBERS INTO A GRID  KM X KM;
C  THEN DOES CHI SQUARE TEST
C
      INTEGER*4 JSEED(23), KSEED
      DIMENSION KARY(100,100)
C
      DATA JSEED / 0, 1, 2, 3, 4, 123456789, 1111111, 6999,
     &          65536, 16777216, 1078741824, 10, 100, 194305786,
     &         1217344457, 314159276, 543219876, 9571916, 5868958,
     &         1234567890, 1933985544, 2050954260, 918807827  /
C
      LIM = 20000
C     LIM = 300
      KM = 64
C     KM = 10
C
      WRITE(3,29) LIM, KM, KM
   29 FORMAT('PAIR UNIFORMITY CHI-SQUARE TEST FOR GENERATOR: GEN_J'/
     & I8,' PAIRS, SORTED INTO',I4,' BY',I4,' GRID'/
     & 'SEED       VALUE CHISQUARE   P VALUE')
C
      DO 300 NS = 9, 10
      KSEED = JSEED(NS)
C
      DO 40 J = 1, KM
        DO 30 I = 1, KM
          KARY(I,J) = 0
   30   CONTINUE
   40 CONTINUE
C
      DO 90 L = 1, LIM
        X1 = GEN_J(KSEED)
        X2 = GEN_J(KSEED)
C       WRITE(3, 49) X1, X2
   49   FORMAT(2F10.6)
        I = KM * X1 + 1
        J = KM * X2 + 1
        KARY(I,J) = KARY(I,J) + 1
C
   90 CONTINUE
C
      SUM = 0.
      DO 190 J = 1, KM
        DO 160 I = 1, KM
          SUM = SUM + KARY(I,J)**2
  160   CONTINUE
C
  190 CONTINUE
C
      CHISQ = SUM*KM*KM/LIM - LIM
      X1 = SQRT(2.*CHISQ) - SQRT(2.*KM*KM - 3.)
      PVAL = 1. - GAUSPRB(X1)
C
      DO 285 J = 1, KM
```

```
      WRITE(3,269) (KARY(I,J),I=1,KM)
  269  FORMAT(6413)
  285 CONTINUE
C
     WRITE(3,299) NS, JSEED(NS), CHISQ, PVAL
  299 FORMAT(I4,I12,F10.2,F10.6)
C
  300 CONTINUE
     STOP 'CHI-SQUARE TEST COMPLETED'
     END
*******************************************************************************
     PROGRAM RANTEST4
C
C  GAP TEST.  FROM KNUTH, VOL 2, p 56
C
     INTEGER*4 JSEED(23), KSEED
     REAL*8 GEN_F
     DIMENSION KARY(5,100), ALPH(5), KT(5), PVAL(5), ISUM(5), CHISQ(5)
     LOGICAL DEBUG
C
     DATA JSEED / 0, 1, 2, 3, 4, 123456789, 1111111, 6999,
    &          65536, 16777216, 1078741824, 10, 100, 194305786,
    &         1217344457, 314159276, 543219876, 9571916, 5868958,
    &         1234567890, 1933985544, 2050954260, 918807827  /
C
     DATA NGAP / 7000 /, MT / 70 /, BETA / .1 /, LIM / 100000 /
     DATA ALPH / .0,  .15,  .45,  .75,  .90 /
     DEBUG = .FALSE.
C
     WRITE(3,29) NGAP, MT, BETA, (ALPH(I), I=1,5)
   29 FORMAT('GAP CHI-SQUARE TEST FOR GENERATOR: GEN_F'/
    & I8,' GAPS OF UP TO LENGTH',I4,' OF WIDTH',F6.4/
    & 'SEED       VALUE CHISQUARE   Ps',F5.2,4F10.2)
C
C  DO FOR EACH SEED
C
     DO 300 NS = 2, 23
     KSEED = JSEED(NS)
C
C  CLEAR COUNTING ARRAYS
C
     DO 40 J = 1, MT
       DO 30 I = 1, 5
         KARY(I,J) = 0
   30  CONTINUE
   40 CONTINUE
     DO 50 I = 1, 5
       KT(I) = 1
       ISUM(I) = 1
   50 CONTINUE
     NDRAW = 0
C
C  PROCESS  NGAP  CASES WHERE GAP IS BETWEEN ALPH AND BETA
C
     DO 140 L = 1, LIM
       X1 = GEN_F(KSEED)
       NDRAW = NDRAW + 1
C        IF (DEBUG) WRITE(3,59) X1
   59   FORMAT(F10.6)
       DO 100 I = 1, 5
         IF (X1 .LT. ALPH(I)) GO TO 120
         IF (X1 .GT. ALPH(I) + BETA) GO TO 100
         K = KT(I)
         IF (ISUM(I) .LE. NGAP) THEN
           KARY(I,K) = KARY(I,K) + 1
```

```
          KT(I) = 0
          ISUM(I) = ISUM(I) + 1
          GO TO 120
        END IF
  100   CONTINUE
C
C  INCREASE EACH GAP LENGTH COUNTER
C
  120 CONTINUE
C
      NQUIT = 0
      DO 130 I = 1, 5
        KT(I) = MIN(KT(I) + 1, MT)
C
C  ARE ALL GAP COUNTS COMPLETE?
C
      IF (ISUM(I) .GT. NGAP) NQUIT = NQUIT + 1
  130   CONTINUE
    IF (NQUIT .GE. 5) GO TO 160
  140 CONTINUE
C
C  IF DROP THROUGH LOOP, THEN DRAWING  LIM  NUMBERS WAS NOT
C  ENOUGH TO GET  NGAP  GAPS.  THE GENERATOR MUST BE DOING
C  BADLY.  SKIP CHI-SQUARE
C
    WRITE(3, 149) (ISUM(I)-1, I=1, 5)
  149 FORMAT('FAILURE TO COMPLETE.  # GAPS RECORDED:',5I5)
C
    IF (DEBUG) THEN
      DO 150 I = 1, 5
        WRITE(3,169) (KARY(I,J), J=1,70)
  150   CONTINUE
    END IF
C
    GO TO 300
C
C  FORM CHI SQUARED STATISTIC
C
  160 CONTINUE
C
    IF (DEBUG) THEN
      DO 170 I = 1, 5
        WRITE(3,169) (KARY(I,J), J=1,70)
  169     FORMAT(20I4)
  170   CONTINUE
    END IF
C
    Q = 1. - BETA
    DO 200 I = 1, 5
      SUM = 0.
      NSUM = 0
      DO 190 J = 1, MT - 1
        SUM = SUM + KARY(I,J)**2 / (BETA * Q**(J-1))
        NSUM = NSUM + KARY(I,J)
  190   CONTINUE
C
      SUM = SUM + KARY(I,MT)**2 / (Q**(MT-1))
      NSUM = NSUM + KARY(I,MT)
C
      CHISQ(I) = SUM/NGAP - NGAP
C       X1 = SQRT(2.*CHISQ(I)) - SQRT(2.*MT - 3.)
C       PVAL = 1. - GAUSPRB(X1)
C
C     IF (CHISQ(I) .GE. 2.*MT) THEN
      IF (DEBUG) THEN
```

```
        WRITE(3,199) I, SUM, NSUM, CHISQ(I)
 199    FORMAT(I4,F10.4,I4,F10.1)
      END IF
C     END IF
C
      NU = MT - 1
      X2 = ((CHISQ(I)/NU)**.333 - (1. - 2./9./NU)) / SQRT(2./9./NU)
      PVAL(I) = 1. - GAUSPRB(X2)
 200 CONTINUE
C
    WRITE(3,249) NS, JSEED(NS), CHISQ(1), (PVAL(I),I=1,5), NDRAW
 249 FORMAT(I4,I12,F10.2,5F10.6,I12)
    IF (DEBUG) WRITE(3,259) NDRAW, NSUM
 259 FORMAT('PVAL, NDRAW, SUM2',2I10)
C
 300 CONTINUE
    STOP 'CHI-SQUARE TEST COMPLETED'
    END
*******************************************************************************
    PROGRAM RANTEST5
C
C PERMUTATION TEST
C CATEGORIZES SETS OF SIX NUMBERS BY THEIR ORDER, THEN DOES CHI SQUARE TEST
C
    REAL*8 GEN_D
    INTEGER*4 JSEED(23), KSEED
    DIMENSION KARY(720), IC(6), UN(6)
    LOGICAL DEBUG
C
    DATA JSEED / 0, 1, 2, 3, 4, 123456789, 1111111, 6999,
   &          65536, 16777216, 1078741824, 10, 100, 194305786,
   &        1217344457, 314159276, 543219876, 9571916, 5868958,
   &        1234567890, 1933985544, 2050954260, 918807827  /
C
    LIM = 5000
    KM = 6
    KFAC = 720
C
    DEBUG = .FALSE.
    IF (DEBUG) THEN
      LIM = 100
      KM = 4
      KFAC = 24
    END IF
C
    WRITE(3,29) LIM, KM, KFAC
  29 FORMAT('PERMUTATION CHI-SQUARE TEST FOR GENERATOR: GEN_D'/
   & I6,' SETS OF',I2,' SORTED BY WHICH OF',I4,' PERMUTATIONS'/
   & 'SEED       VALUE CHISQUARE   P VALUE')
C
    DO 300 NS = 1, 23
    KSEED = JSEED(NS)
C
    DO 40 J = 1, KFAC
      KARY(J) = 0
  40 CONTINUE
    CHISQ = 0.
    PVAL = 0.
C
    DO 190 L = 1, LIM
      DO 80 J = 1, KM
        UN(J) = GEN_D(KSEED)
  80  CONTINUE
C       IF (UN(KM) .EQ. UN(KM-1)) GO TO 250
C
```

```
      IF (DEBUG) THEN
       WRITE(3,89) (UN(I),I=1,KM)
   89   FORMAT(6X,6F10.6)
      END IF
C
C  SORT TO DETERMINE PERMUTATION NUMBER
C
      DO 130 JR = KM, 1, -1
       UMAX = 0.
       DO 110 I = 1, JR
        IF (UN(I) .LT. UMAX) GO TO 110
        UMAX = UN(I)
        IMAX = I
  110   CONTINUE
C
       IC(JR) = IMAX - 1
       SWAP = UN(JR)
       UN(JR) = UN(IMAX)
       UN(IMAX) = SWAP
  130  CONTINUE
C
C  USE KNUTH FORMULA TO IDENTIFY PERMUTATION NUMBER
C
      ISUM = 0
      DO 150 I = 1, KM-1
       ISUM = ISUM + IC(I)
       ISUM = ISUM * (I+1)
  150  CONTINUE
      ISUM = ISUM + IC(KM) + 1
C
      IF (DEBUG) THEN
       WRITE(3,159) ISUM
  159   FORMAT(I6)
      END IF
C
      IF (ISUM .LE. 0 .OR. ISUM .GT. KFAC) THEN
       WRITE(3,169) NS, L, ISUM
  169   FORMAT('INDEX ERROR FOR SEED, SET, VALUE',3I4)
      ELSE
       KARY(ISUM) = KARY(ISUM) + 1
      END IF
C
  190 CONTINUE
C
     SUM = 0.
     ISUM2 = 0
     DO 210 I = 1, KFAC
      SUM = SUM + KARY(I)**2
      ISUM2 = ISUM2 + KARY(I)
  210 CONTINUE
C
     CHISQ = SUM*KFAC/LIM - LIM
C    IF (CHISQ .LE. 0.) THEN
C     IF (DEBUG) THEN
      WRITE(3,239) (KARY(I), I=1,KFAC)
  239  FORMAT(13I6)
C      PVAL = 1.
C      GO TO 250
C     END IF
C
     X1 = SQRT(2.*CHISQ) - SQRT(2.*KFAC - 3.)
     PVAL = 1. - GAUSPRB(X1)
C
  250 CONTINUE
     WRITE(3,299) NS, JSEED(NS), CHISQ, PVAL
```

```
 299 FORMAT(I4,I12,F10.2,F10.6)
C
 300 CONTINUE
     STOP 'CHI-SQUARE TEST COMPLETED'
     END
*******************************************************************************
     PROGRAM RANTEST6
C
C  RUN TEST.  FROM KNUTH, VOL 2, p 68
C
     INTEGER*4 JSEED(23), KSEED
     REAL*8 GEN_D
     DIMENSION KARY(100)
     LOGICAL DEBUG
C
     DATA JSEED / 0, 1, 2, 3, 4, 123456789, 1111111, 6999,
    &          65536, 16777216, 1078741824, 10, 100, 194305786,
    &         1217344457, 314159276, 543219876, 9571916, 5868958,
    &         1234567890, 1933985544, 2050954260, 918807827  /
C
     DATA NRUN / 25000 /, MT / 6 /, PI / 3.14159276 /
     DEBUG = .FALSE.
C
     WRITE(3,29) NRUN, MT
  29 FORMAT('RUN CHI-SQUARE TEST FOR GENERATOR: GEN_D'/
    & I8,' RUNS OF OF INCREASING VALUES UP TO LENGTH',I4/
    & 'SEED       VALUE CHISQUARE   P VALUE')
C
C  DO FOR EACH SEED
C
     DO 300 NS = 2, 23
     KSEED = JSEED(NS)
C
C  CLEAR KOUNTING ARRAY
C
     DO 40 J = 1, MT
       KARY(J) = 0
  40 CONTINUE
     NDRAW = 0
     CHISQ = 0.
     PVAL = 0.
C
C  PROCESS  NRUN  CASES
C
     DO 140 L = 1, NRUN
       X0 = GEN_D(KSEED)
C       IF (DEBUG) WRITE(3,69) X0
  69   FORMAT(F10.6)
       NDRAW = NDRAW + 1
       DO 100 J = 1, MT - 1
         X1 = GEN_D(KSEED)
C          IF (DEBUG) WRITE(3,69) X1
         NDRAW = NDRAW + 1
         IF (X1 .LT. X0) GO TO 120
         IF (X1 .EQ. X0) THEN
           WRITE(*,79) X1, KSEED, L, J
           WRITE(3,79) X1, KSEED, L, J
  79       FORMAT(' SUCCESSIVE X''s EQUAL TO',F15.12,
    &            ' SEED AND INDICES',I12, 2I6)
           GO TO 120
         END IF
         X0 = X1
 100   CONTINUE
C
C  IF FALL THROUGH LOOP, RUN IS AT LEAST  MT
```

```
C
      KARY(MT) = KARY(MT) + 1
      GO TO 140
C
  120 CONTINUE
C
      KARY(J) = KARY(J) + 1
  140 CONTINUE
C
     IF (DEBUG) THEN
       WRITE(3,159) (KARY(I), I=1, MT)
  159  FORMAT(10I5)
     END IF
C
C  FORM CHI SQUARED STATISTIC
C
     SUM = 0.
     NSUM = 0
     FAC = 1.
     DO 190 J = 1, MT - 1
       FAC = FAC * (J + 1.)
       P = J / FAC
       SUM = SUM + KARY(J)**2 / P
       NSUM = NSUM + KARY(J)
  190 CONTINUE
C
     P = 1. / FAC
     SUM = SUM + KARY(MT)**2 / P
     NSUM = NSUM + KARY(MT)
C
     CHISQ = SUM/NRUN - NRUN
C      IF (CHISQ .LE. 0.) THEN
C        PVAL1 = 1.
C        IF (DEBUG) THEN
C          WRITE(3,199) (KARY(I),I=1,MT)
C        END IF
C        GO TO 280
C      END IF
     X1 = SQRT(2.*CHISQ) - SQRT(2.*MT - 3.)
     PVAL1 = 1. - GAUSPRB(X1)
C
     IF (CHISQ .GE. 2.*MT) THEN
       IF (DEBUG) THEN
         WRITE(3,199) (KARY(I),I=1,MT)
  199    FORMAT(20I4)
       END IF
     END IF
C
     NU = MT - 1
     X2 = ((CHISQ/NU)**.333 - (1. - 2./9./NU)) / SQRT(2./9./NU)
     PVAL2 = 1. - GAUSPRB(X2)
C
C  SMALL NU EXPANSION, FOR NU ODD.  ABRAMOWITZ & STEGUN p 941
C
     CHI = SQRT(CHISQ)
     LIMR = (NU - 1) / 2
     RESULT = 0.
     DO 230 IR = LIMR, 1, -1
       RESULT = RESULT + 1.
       F = CHISQ / (2.*IR - 1.)
       RESULT = RESULT * F
  230 CONTINUE
     RESULT = RESULT * EXP(-CHISQ/2.) / SQRT(2.*PI)
     PVAL = 2.*(1. - GAUSPRB(CHI) + RESULT / CHI)
C
```

```
C  PRINT LINE OF OUTPUT TABLE FOR THIS SEED
C
  280 CONTINUE
      WRITE(3,299) NS, JSEED(NS), CHISQ, PVAL, NDRAW
  299 FORMAT(I4,I12,F10.2,F10.6,I12)
      IF (DEBUG) WRITE(3,309) PVAL1, PVAL2, NDRAW, NSUM, RESULT, F
  309 FORMAT('PVALs, NDRAW, SUM2',2F10.6,2I10,F10.6,F10.2)
C
  300 CONTINUE
      STOP 'RUN TEST COMPLETED'
      END
*******************************************************************************
      PROGRAM RANTEST7
C
C  MARSAGLIA OVERLAPPING TRIPLES TEST
C  USES 30,000 NUMBERS AND LOOKS AT TRIPLES.  UNIFORMITY AND INDEPENDENCE
C
      INTEGER*4 JSEED(23), KSEED
      DIMENSION KARY(12,12,12), LARY(12,12)
C
      DATA JSEED / 0, 1, 2, 3, 4, 123456789, 1111111, 6999,
     &          65536, 16777216, 1078741824, 10, 100, 194305786,
     &         1217344457, 314159276, 543219876, 9571916, 5868958,
     &         1234567890, 1933985544, 2050954260, 918807827  /
C
      LIM = 30000
      KM = 8
C
      WRITE(3,29) LIM, KM, KM, KM
   29 FORMAT('OVERLAPPING TRIPLES TEST FOR GENERATOR: GEN_K'/
     & I8,' SETS, SORTED INTO',I3,' BY',I3,' BY',I3,' GRID'/
     & 'SEED       VALUE CHISQUARE   P VALUE')
C
      DO 300 NS = 1, 23
      KSEED = JSEED(NS)
      CHISQ = 0.
      PVAL = 0.
C
      DO 40 K = 1, KM
        DO 30 J = 1, KM
          LARY(J,K) = 0
          DO 20 I = 1, KM
            KARY(I,J,K) = 0
   20     CONTINUE
   30   CONTINUE
   40 CONTINUE
C
      X1 = GEN_K(KSEED)
      X2 = GEN_K(KSEED)
C       WRITE(3,49) X1*KM
C       WRITE(3,49) X2*KM
C   49 FORMAT(F10.6)
      IF (X1 .EQ. X2) GO TO 280
C
      I0 = KM * X1 + 1
      J0 = KM * X2 + 1
C
C  SORT SUCCESSIVE TRIPLES FROM X1, X2, X3
C
      DO 90 L = 3, LIM
        X = GEN_K(KSEED)
C         WRITE(3,49) X*KM
        K0 = KM * X + 1
        KARY(I0,J0,K0) = KARY(I0,J0,K0) + 1
        LARY(J0,K0) = LARY(J0,K0) + 1
```

```
      I0 = J0
      J0 = K0
   90 CONTINUE
C
C  ADD TRIPLE Xn-1, Xn, X1
C
      K0 = KM * X1 + 1
      KARY(I0,J0,K0) = KARY(I0,J0,K0) + 1
      LARY(J0,K0) = LARY(J0,K0) + 1
      I0 = J0
      J0 = K0
C
C  FINALLY Xn, X1, X2 TO COMPLETE CYCLE
C
      K0 = KM * X2 + 1
      KARY(I0,J0,K0) = KARY(I0,J0,K0) + 1
      LARY(J0,K0) = LARY(J0,K0) + 1
C
C  FORM CHI-SQUARE PIECES
C
      SUMK = 0.
      SUML = 0.
      DO 190 K = 1, KM
        DO 160 J = 1, KM
          SUML = SUML + LARY(J,K)**2
          DO 130 I = 1, KM
            SUMK = SUMK + KARY(I,J,K)**2
  130     CONTINUE
  160   CONTINUE
  190 CONTINUE
C
C     WRITE(3, 209) ((LARY(J,K),J=1,KM),K=1,KM)
C  209 FORMAT(16I4)
C     DO 220 K = 1, KM
C       WRITE(3,209) ((KARY(I,J,K),I=1,KM),J=1,KM)
C  220 CONTINUE
C
      CHISQ = (SUMK*KM**3 - SUML*KM*KM) / LIM
      NU = KM**3 - KM*KM
      X1 = SQRT(2.*CHISQ) - SQRT(2.*NU - 1.)
      PVAL = 1. - GAUSPRB(X1)
C
  280 CONTINUE
      WRITE(3,299) NS, JSEED(NS), CHISQ, PVAL
  299 FORMAT(I4,I12,F10.2,F10.6)
C
  300 CONTINUE
      STOP 'OVERLAPPING TRIPLES TEST COMPLETED'
      END
*************************************************************************
      FUNCTION GAUSPRB(X)
C
C  COMPUTES THE PROBABILITY THAT A RANDOM VARIABLE  Y  THAT HAS
C  THE GAUSSIAN DISTRIBUTION WITH MEAN  0  AND VARIANCE  1  IS
C  LESS THAN THE INPUT VALUE  X.  ABRAMOWITZ & STEGUN p 932
C
      REAL*8 DCOEFF(6), SUM, XX
      DATA DCOEFF / .0498673470, .0211410061, .0032776263,
     &              .0000380036, .0000488906, .0000053830  /
C
      XX = X
      IF (X .LT. 0.) XX = -X
      SUM = 1.
      DO 50 I = 1, 6
        SUM = SUM + DCOEFF(I) * XX**I
```

```
  50 CONTINUE
C
      XX = 1. - .5 * SUM**(-16)
      GAUSPRB = XX
      IF (X .LT. 0.) GAUSPRB = 1. - XX
C
      RETURN
      END
```

APPENDIX C - RESULTS OF RANDOM NUMBER GENERATOR TESTS

Results Summary

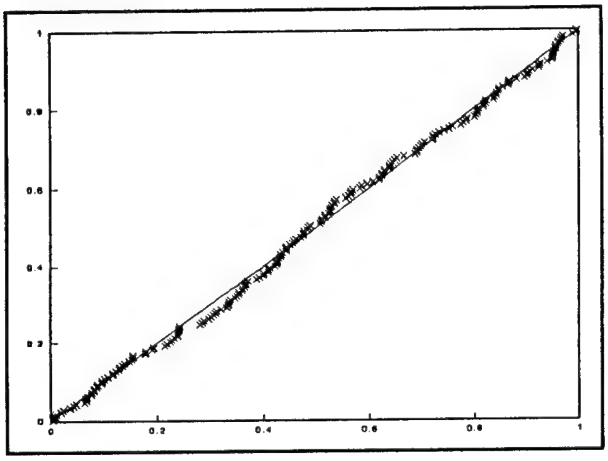
It was noted early in testing that UNIF and UNIF2 are essentially identical, so only the latter was used for most testing. Some generators gave consistently good results (denoted by . in the table), some good results except for specific seeds (seed numbers), some gave slightly bad results for several seeds (?), and some gave consistently bad results (X).

The best generators are GEN_K and GEN_H, followed by GEN_B, GEN_A, and UNIF2. The worst seed is seed 10, which is 2^{24}, closely followed by 9, which is 2^{16}. Large powers of 2 make bad seeds. However, seed 19 is sometimes a bad choice, and that is used because it is a friend's phone number.

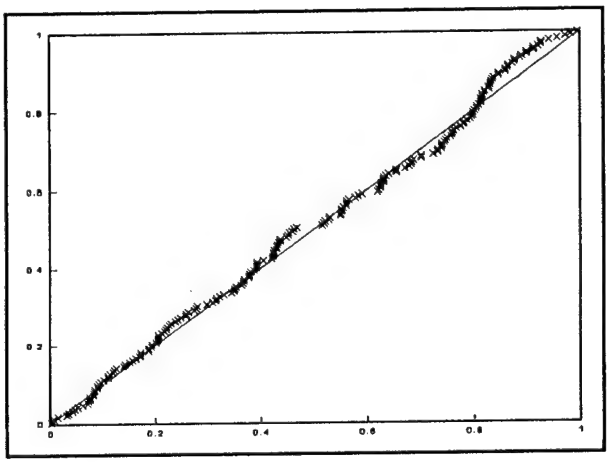
Generator	Rantest3	Rantest4	Rantest5	Rantest6	Rantest7
UNIF2	.	.	?	.	10
GEN_A	.	.	.	?	.
GEN_B	.	.	.	.	10
GEN_C	.	?	.	X	.
GEN_D	X	X	X	X	X
GEN_E	.	X	X	X	X
GEN_F	X	X	X	X	X
GEN_G	X	9,10,19	9,10,11,19	9,10,19	9,10,14
GEN_H	.	.	.	.	.
GEN_I	9,10	9,10	9,10	10	9,10,19
GEN_J	X	9,10	10	10,16	9,10
GEN_K	.	.	.	.	.
GEN_L	X	9,10,22	9,10	X	X

Summary Plots

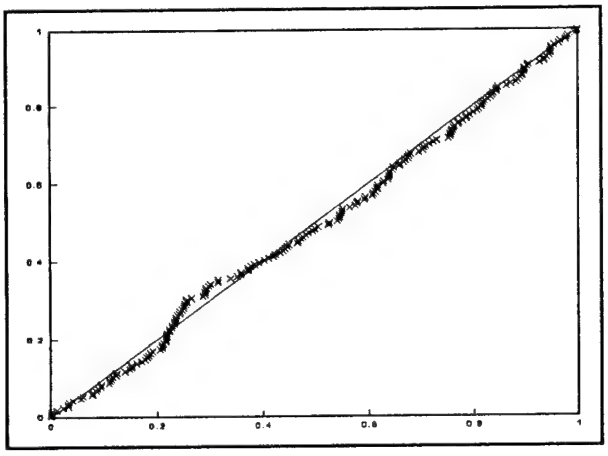
The results of the Rantests are P values from chi-square tests. If a generator is good, then these P values should be uniformly distributed between 0 and 1. Therefore a composite summary of the test results may be made by plotting the sorted P values and comparing to a 45 degree line. The next two pages have such plots for 12 of the generators subjectively ranked in order of decreasing quality. Generators GEN_E and GEN_F are so bad that their plots are not shown; the latter is essentially all zeros.



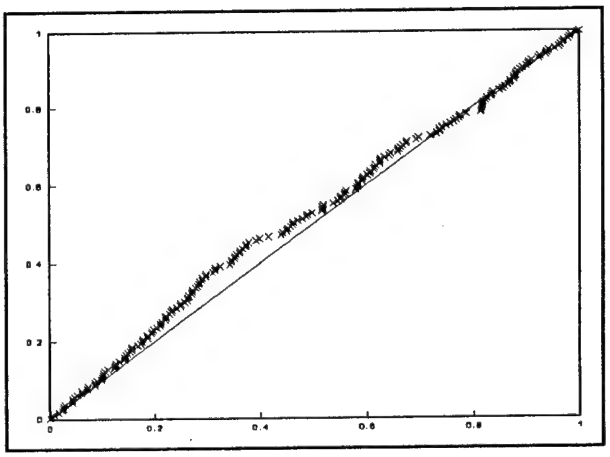
1. GEN_K



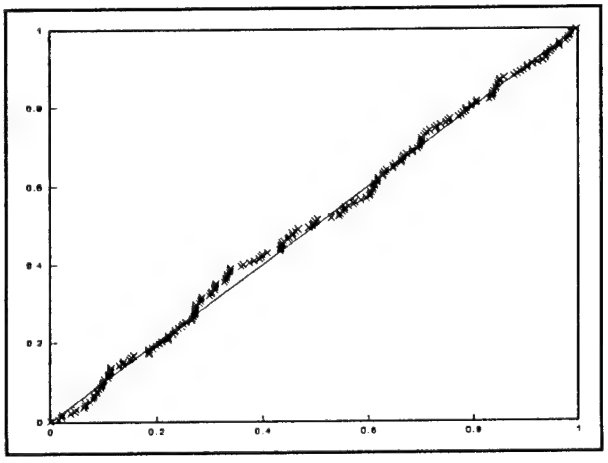
4. GEN_A



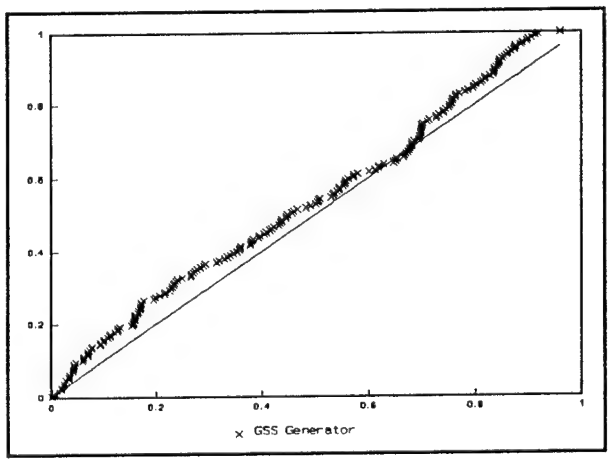
2. GEN_H



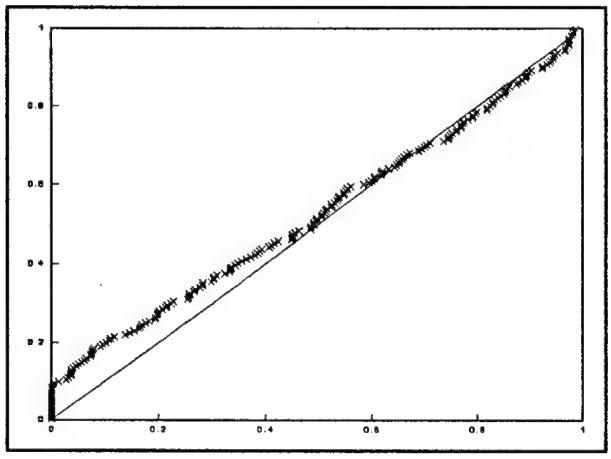
5. UNIF2



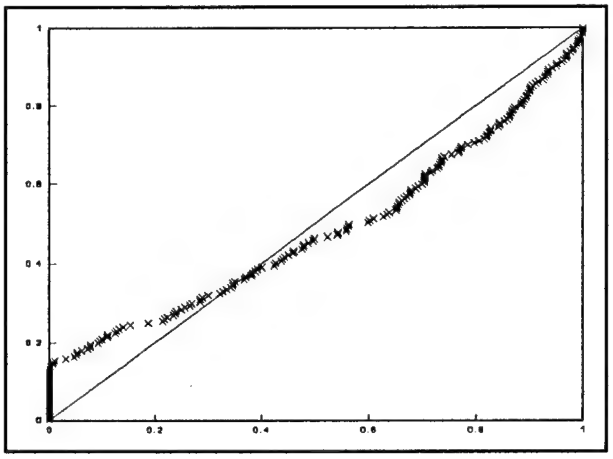
3. GEN_B



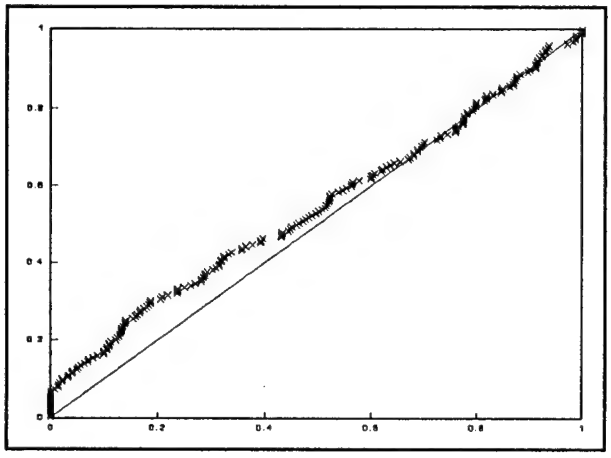
6. GSS GENERATOR



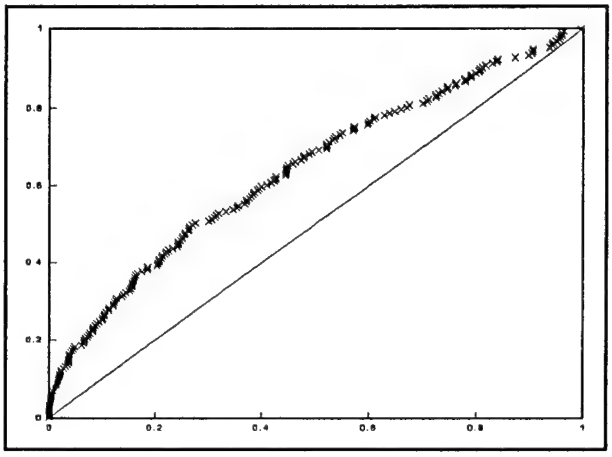
7. GEN_I



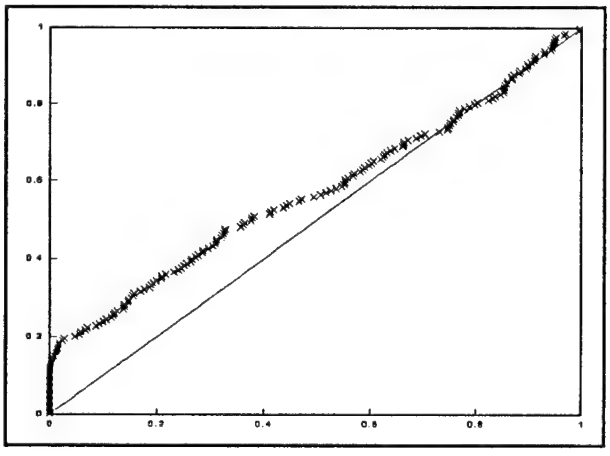
10. GEN_G



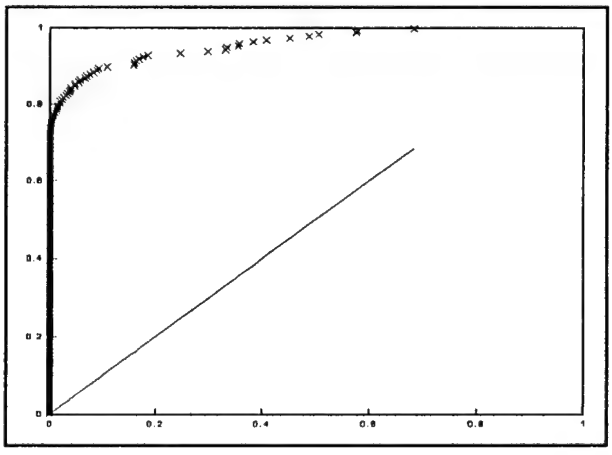
8. GEN_J



11. GEN_C

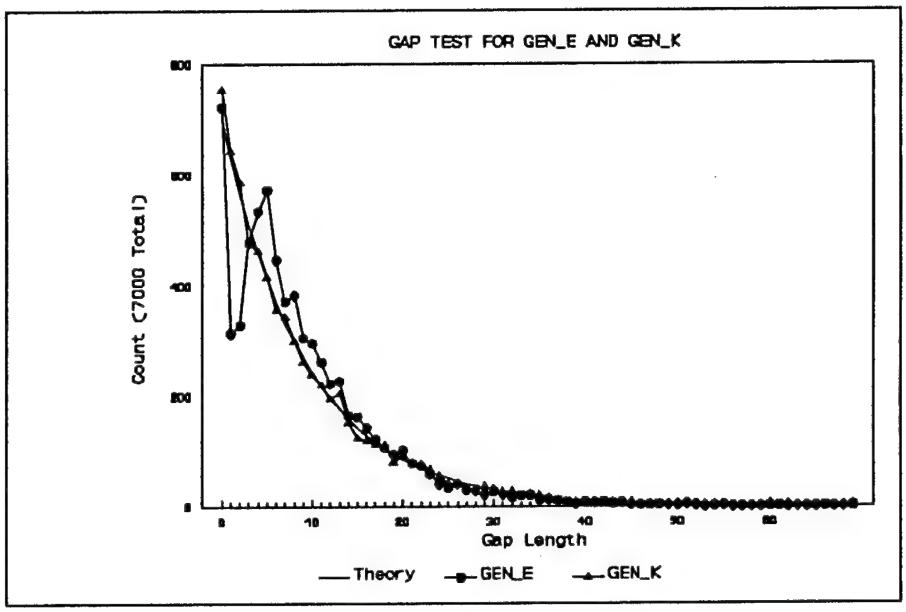


9. GEN_L



12. GEN_D

A Few Detailed Samples



The chi-square values do not indicate what is going on that makes the generators good or bad. Just a few examples will illustrate.

The figure shows the number of gaps of each length from Rantest4 for the good generator GEN_K and for GEN_E (terrible). The theoretical distribution is also shown; it is barely discernable from the results for GEN_K. For GEN_E gaps of lengths 1 and 2 are badly underrepresented, with gaps of 4 and up making up the difference.

Patterns are discernable in the 64 × 64 array computed in Rantest3 for bad generators. Portions of the array are given for two of the generators. A good generator has the entries distributed randomly around 4.88.

Rantest3 Array for GEN_D (portion)

```
17  0 20  0  0 20  0  0 14  0  0 16  0 20  0  0 13  0  0 11  0  0 14  0  0  8  0 17  0  0 18  0
13  0 15  0  0 19  0  0 12  0  0 17  0  0 17  0 18  0  0 13  0  0 14  0  0  9  0 16  0  0 13  0
16  0 17  0  0 15  0  0 21  0  0 22  0  0 11  0 19  0  0 16  0  0 17  0  0 16  0  0 14  0 18  0
19  0  0  9  0 13  0  0  9  0  0  8  0  0 12  0 15  0  0 15  0  0  8  0  0 17  0  0 10  0 13  0
15  0  0 11  0 16  0  0 12  0  0 17  0  0 17  0  0 12  0  7  0  0 16  0  0 15  0  0  7  0 19  0
 9  0  0 14  0  0 13  0 14  0  0 13  0  0 24  0  0 10  0  8  0  0 11  0  0 12  0  0 12  0  0 20
12  0  0 10  0  0 17  0 13  0  0  7  0  0 15  0  0 19  0  0 14  0 16  0  0 15  0  0 12  0  0 15
12  0  0 12  0  0 12  0 17  0  0 17  0  0 16  0  0 16  0  0 13  0 12  0  0 19  0  0  6  0  0 15
16  0  0 12  0  0 19  0  0 13  0  6  0  0  7  0  0  7  0  0 13  0 10  0  0 12  0  0 14  0  0 15
17  0  0  9  0  0 11  0  0 15  0  7  0  0 19  0  0 12  0  0 13  0  0 15  0 12  0  0  8  0  0 14
 9  0  0 14  0  0 12  0  0 18  0  0 15  0  9  0  0 10  0  0 15  0  0 14  0 12  0  0  7  0  0 16
 9  0  0 10  0  0 19  0  0 12  0  0 11  0 19  0  0 12  0  0 17  0  0 12  0  0 11  0 15  0  0 14
 0 17  0  8  0  0 13  0  0 13  0  0 14  0 10  0  0 12  0  0 13  0  0 12  0  0 14  0 12  0  0  8
 0 13  0 17  0  0 13  0  0  8  0  0 16  0  0 14  0 19  0  0 14  0  0 10  0  0 11  0 13  0  0 12
 0 14  0  0 18  0 19  0  0 20  0  0 16  0  0 12  0  9  0  0 14  0  0 19  0  0 12  0  0 15  0 16
```

Rantest3 Array for GEN_F (portion)

```
79142126128 30  0  0  0 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
39108128137102  0  0  0103  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0 95116127124 28  0  0126 33  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0 31127118137 90  0  0128106  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0 93137164121 32  0120136 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0 28118121140 95  0136134 91  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0102126128120 34124119110 40  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0 28128137136 96137118142 80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0 86127124119129121126128 30  0  0  0 35  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0 31118137118 30140128137 96  0  0  0 84  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0100164121  0 99120136124 37  0  0 25 19  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0 32121140  0 20129134119 86  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0 89128  0  0 23112110142 37  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0 29137  0  0  0 20131108 99  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0 99  0  0  0  0 26122116 28  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0 31  0  0  0  0  0 25114 87  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17130  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 22  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Selected Output Files

Given are the output files from tests Rantest3, 5, 4, 6, and 7 on GEN_K, which is a good combination generator based on a suggestion of Marsaglia [1985]. These are followed by files for GEN_L, a moderately bad generator, and finally those for the generator in the GSS system. These are presented here because the P values from these output arrays form the raw data for the plots given on page 22. Another analyst might want to view the data in other ways.

PAIR UNIFORMITY CHI-SQUARE TEST FOR GEN_K
20000 PAIRS, SORTED INTO 64 BY 64 GRID

SEED	VALUE	CHISQUARE	P VALUE
2	1	4129.54	.349621
3	2	4088.58	.526107
4	3	4016.49	.806821
5	4	4088.17	.527910
6	123456789	4120.52	.387005
7	1111111	3944.40	.952956
8	6999	4119.30	.392197
9	65536	4159.03	.238767
10	16777216	4012.39	.819103
11	1078741824	4103.32	.461192
12	10	4128.72	.352965
13	100	4213.09	.096598
14	194305786	4040.24	.726269
15	1217344457	4195.07	.134659
16	314159276	4041.88	.720169
17	543219876	4055.81	.665887
18	9571916	4189.75	.147669
19	5868958	4158.62	.240161
20	1234567890	3980.85	.897014
21	1933985544	4064.82	.628766
22	2050954260	4187.29	.153951
23	918807827	3999.28	.855061

PERMUTATION CHI-SQUARE TEST FOR GEN_K
5000 SETS OF 6 SORTED BY 720 PERMUTATIONS

SEED	VALUE	CHISQUARE	P VALUE
1	0	657.18	.950892
2	1	698.94	.698274
3	2	770.66	.088225
4	3	620.90	.996193
5	4	657.47	.950080
6	123456789	746.46	.232499
7	1111111	684.54	.817689
8	6999	719.97	.484562
9	65536	737.82	.306295
10	16777216	776.42	.067040
11	1078741824	724.00	.442430
12	10	727.17	.409833
13	100	668.13	.911964
14	194305786	720.83	.475494
15	1217344457	751.65	.193635
16	314159276	815.30	.006703
17	543219876	660.64	.940427
18	9571916	711.90	.569225
19	5868958	799.74	.018498
20	1234567890	664.96	.925026
21	1933985544	712.77	.560214
22	2050954260	738.40	.301053
23	918807827	700.10	.687453

GAP CHI-SQUARE TEST FOR GENERATOR: GEN_K
7000 GAPS OF UP TO LENGTH 70 OF WIDTH .1000

SEED	VALUE	CHISQUARE	Ps .00	.15	.45	.75	.90
2	1	79.10	.190226	.296844	.010474	.470724	.515326
3	2	87.74	.063665	.411352	.786929	.516364	.360382
4	3	57.26	.842390	.989288	.116408	.408702	.309663
5	4	72.62	.359802	.867519	.522792	.081640	.534074
6	123456789	77.53	.225390	.623744	.004310	.690087	.758689
7	1111111	77.43	.227855	.353782	.076969	.241328	.077465
8	6999	73.94	.320128	.432911	.966023	.238756	.129351
9	65536	75.20	.284696	.774559	.751929	.287742	.628358
10	16777216	68.69	.488021	.901417	.801399	.365851	.107383
11	1078741824	69.05	.475670	.370501	.967465	.101099	.424802
12	10	72.40	.366329	.008136	.823075	.533858	.957262
13	100	56.25	.864898	.722301	.152411	.400661	.117182
14	194305786	50.73	.951373	.903558	.639021	.399486	.620159
15	1217344457	87.19	.068808	.619446	.694625	.475959	.955834
16	314159276	91.43	.036866	.906244	.241722	.640814	.510335
17	543219876	72.46	.364577	.838759	.331026	.346410	.803441
18	9571916	58.10	.822211	.700262	.138668	.842708	.430046
19	5868958	68.06	.509298	.970128	.456839	.128525	.808674
20	1234567890	85.97	.081425	.837129	.334106	.312082	.524949
21	1933985544	62.22	.705412	.482313	.566077	.425684	.454886
22	2050954260	77.83	.218429	.122242	.648191	.421655	.047821
23	918807827	84.78	.095439	.638449	.443233	.043327	.178650

RUN CHI-SQUARE TEST FOR GENERATOR: GEN_K
40000 RUNS OF INCREASING VALUES TO LENGTH 8

SEED	VALUE	CHISQUARE	P VALUE
1	0	7.94	.338271
2	1	2.07	.955841
3	2	5.13	.644541
4	3	3.04	.880977
5	4	5.43	.607205
6	123456789	4.39	.733872
7	1111111	6.02	.537882
8	6999	4.31	.743412
9	65536	3.95	.785289
10	16777216	1.98	.961069
11	1078741824	2.54	.924239
12	10	2.50	.927157
13	100	5.53	.596001
14	194305786	16.26	.022825
15	1217344457	5.64	.582164
16	314159276	5.79	.564615
17	543219876	7.03	.426053
18	9571916	4.75	.690977
19	5868958	4.00	.779778
20	1234567890	6.93	.436078
21	1933985544	6.09	.529327
22	2050954260	3.38	.847850
23	918807827	6.28	.506916

OVERLAPPING TRIPLES TEST FOR GEN_K
30000 SETS, SORTED INTO 8 BY 8 BY 8 GRID

SEED	VALUE	CHISQUARE	P VALUE
1	0	369.25	.996938
2	1	438.07	.624317
3	2	414.93	.866348
4	3	460.28	.335723
5	4	475.61	.177443
6	123456789	435.83	.652697
7	1111111	421.93	.807140
8	6999	451.29	.449650
9	65536	471.43	.214934
10	16777216	468.65	.242287
11	1078741824	489.13	.086795
12	10	465.02	.280911
13	100	493.50	.066808
14	194305786	478.44	.154574
15	1217344457	395.88	.962317
16	314159276	451.81	.442914
17	543219876	443.33	.555527
18	9571916	460.44	.333828
19	5868958	417.44	.846689
20	1234567890	399.90	.949070
21	1933985544	441.02	.586052
22	2050954260	452.65	.431791
23	918807827	430.07	.721902

The following are output files for the generator GEN_L, which uses the multiplier and modulus of RANDU. This is known to be a rather poor generator.

PAIR UNIFORMITY CHI-SQUARE TEST FOR GEN_L
20000 PAIRS, SORTED INTO 64 BY 64 GRID

SEED	VALUE	CHISQUARE	P VALUE
2	1	4149.61	.271951
3	2	4086.53	.535116
4	3	4211.46	.099688
5	4	4136.50	.321707
6	123456789	3999.69	.854015
7	1111111	3987.40	.883233
8	6999	4056.63	.662570
9	65536	406954.30	.000000
10	167772165	100000.00	.000000
11	1078741824	4033.28	.751451
12	10	4288.46	.017063
13	100	4126.67	.361374
14	194305786	4093.08	.506253
15	1217344457	4075.06	.585150
16	314159276	3946.04	.951116
17	543219876	4034.92	.745637
18	9571916	3951.36	.944739
19	5868958	3948.90	.947759
20	1234567890	4139.37	.310495
21	1933985544	4148.38	.276446
22	2050954260	4156.98	.245784
23	918807827	4189.75	.147669

PERMUTATION CHI-SQUARE TEST FOR GEN_L
5000 SETS OF 6 SORTED BY 720 PERMUTATIONS

SEED	VALUE	CHISQUARE	P VALUE
1	0	.00	.000000
2	1	719.10	.493645
3	2	779.30	.058065
4	3	737.54	.308931
5	4	731.78	.363689
6	123456789	723.42	.448413
7	1111111	710.75	.581192
8	6999	682.53	.831754
9	65536	10827.90	.000000
10	16777216	276250.40	.000000
11	1078741824	625.79	.994298
12	10	680.51	.845119
13	100	757.41	.155565
14	194305786	757.98	.152057
15	1217344457	754.24	.175831
16	314159276	783.33	.047142
17	543219876	668.70	.909420
18	9571916	701.82	.670934
19	5868958	663.81	.929398
20	1234567890	707.01	.619580
21	1933985544	762.59	.125935
22	2050954260	705.86	.631197
23	918807827	708.74	.601971

The gap test has an escape provision. If the generator is doing so poorly that the number of draws becomes excessive without reaching the desired 7000 gaps, it quits and prints a message (seed 10 below). If this is the case, then the results should be interpreted as the equivalent of a 0 p value for the Chi squared statistic.

GAP CHI-SQUARE TEST FOR GENERATOR: GEN_L
7000 GAPS OF UP TO LENGTH 70 OF WIDTH .1000

SEED	VALUE	CHISQUARE	Ps .00	.15	.45	.75	.90
2	1	61.41	.730228	.001747	.121468	.912255	.324759
3	2	73.69	.327474	.823231	.910745	.311673	.014619
4	3	76.47	.251395	.765806	.146500	.105356	.857395
5	4	54.58	.897385	.118879	.055237	.852356	.893809
6	123456789	77.91	.216622	.623609	.552809	.013676	.285429
7	1111111	60.33	.762268	.767973	.749646	.066800	.525341
8	6999	77.25	.232011	.903373	.756201	.801640	.466001
9	65536	5645.16	.000000	.000000	.000000	.000000	.000000
FAILURE TO COMPLETE. # GAPS RECORDED: 7000 6250 7000 7000 6250							
11	1078741824	93.36	.027233	.567058	.867746	.662920	.024357
12	10	63.62	.660424	.189541	.255889	.927902	.377883
13	100	51.62	.941418	.597304	.546289	.834878	.626399
14	194305786	50.98	.948734	.411741	.320211	.588686	.850909
15	1217344457	59.07	.797237	.381786	.263738	.695407	.516442
16	314159276	66.86	.550492	.966239	.296031	.016780	.410536
17	543219876	78.25	.208762	.757463	.968911	.240061	.208865
18	9571916	54.59	.897131	.867280	.304678	.880626	.198900
19	5868958	75.93	.265259	.435953	.441941	.327289	.556500
20	1234567890	64.30	.637762	.871017	.645476	.138313	.186125
21	1933985544	41.64	.996213	.605665	.686627	.139398	.777165
22	2050954260	51.10	.947363	.000040	.086565	.070286	.943420
23	918807827	83.32	.115354	.356594	.858891	.092635	.006954

RUN CHI-SQUARE TEST FOR GENERATOR: GEN_L
40000 RUNS OF INCREASING VALUES TO LENGTH 8

SEED	VALUE	CHISQUARE	P VALUE
1	0	.00	.000000
2	1	20.77	.004129
3	2	10.98	.139669
4	3	4.11	.766791
5	4	8.56	.285555
6	123456789	17.91	.012381
7	1111111	7.52	.376506
8	6999	21.15	.003561
9	65536	17609.20	.000000
10	16777216	335832.00	.000000
11	1078741824	9.92	.193152
12	10	17.87	.012591
13	100	3.32	.853419
14	194305786	25.15	.000715
15	1217344457	6.61	.470451
16	314159276	10.35	.169820
17	543219876	7.11	.417804
18	9571916	4.28	.746452
19	5868958	10.59	.157744
20	1234567890	5.89	.552132
21	1933985544	3.94	.786753
22	2050954260	5.75	.568797
23	918807827	4.65	.703024

OVERLAPPING TRIPLES TEST FOR GEN_L
30000 SETS, SORTED INTO 8 BY 8 BY 8 GRID

SEED	VALUE	CHISQUARE	P VALUE
2	1	2027.60	.000000
3	2	1076.94	.000000
4	3	1962.59	.000000
5	4	523.79	.007148
6	123456789	569.36	.000065
7	1111111	1126.67	.000000
8	6999	1938.12	.000000
9	65536	133342.20	.000000
10	16777216	389944.10	.000000
11	1078741824	1149.43	.000000
12	10	599.48	.000001
13	100	1178.56	.000000
14	194305786	1127.78	.000000
15	1217344457	14886.00	.000000
16	314159276	1962.35	.000000
17	543219876	2013.99	.000000
18	9571916	599.09	.000001
19	5868958	555.20	.000330
20	1234567890	1145.26	.000000
21	1933985544	1024.75	.000000
22	2050954260	14852.44	.000000
23	918807827	1065.40	.000000

The following files are test results from the generator used in the GSS system at CECOM, and therefore in the SPM.

PAIR UNIFORMITY CHI-SQUARE TEST FOR GSS
20000 PAIRS, SORTED INTO 64 BY 64 GRID

SEED	VALUE	CHISQUARE	P VALUE
1	0	4052.12	0.680654
2	1	4134.45	0.329821
3	2	4146.33	0.284018
4	3	4118.89	0.393931
5	4	4113.15	0.418440
6	123456789	4116.43	0.404385
7	1111111	3999.28	0.855061
8	6999	4018.12	0.801770
9	65536	4115.61	0.407892
10	16777216	4015.26	0.810557
11	1078741824	4047.21	0.699907
12	10	4112.33	0.421977
13	100	3993.96	0.868204
14	194305786	4003.79	0.843279
15	1217344457	4047.21	0.699907
16	314159276	3979.62	0.899461
17	543219876	3996.42	0.862243
18	9571916	4086.94	0.533317
19	5868958	3997.64	0.859196
20	1234567890	4047.21	0.699907
21	1933985544	4160.26	0.234611
22	2050954260	4121.34	0.383558
23	918807827	4128.72	0.352965

PERMUTATION CHI-SQUARE TEST FOR GENERATOR IN GSS
5000 SETS OF 6 SORTED BY WHICH OF 720 PERMUTATIONS

SEED	VALUE	CHISQUARE	P VALUE
1	0	765.18	0.112730
2	1	775.55	0.069936
3	2	775.55	0.069936
4	3	775.55	0.069936
5	4	777.86	0.062425
6	123456789	723.71	0.445420
7	1111111	724.58	0.436460
8	6999	730.05	0.380787
9	65536	702.40	0.665356
10	16777216	736.96	0.314234
11	1078741824	713.06	0.557203
12	10	778.14	0.061533
13	100	763.46	0.121416
14	194305786	784.77	0.043671
15	1217344457	714.21	0.545140
16	314159276	755.10	0.170141
17	543219876	794.85	0.024811
18	9571916	714.21	0.545140
19	5868958	756.54	0.160929
20	1234567890	695.20	0.732275
21	1933985544	711.62	0.572220
22	2050954260	707.01	0.619583
23	918807827	756.83	0.159127

GAP CHI-SQUARE TEST FOR GENERATOR IN GSS
6500 GAPS OF UP TO LENGTH 70 OF WIDTH 0.1000

SEED	VALUE	CHISQUARE	Ps 0.00	0.15	0.45	0.75	0.90
1	0	60.06	0.770141	0.750811	0.270583	0.233189	0.628845
2	1	59.66	0.781346	0.650550	0.019477	0.817679	0.676581
3	2	59.10	0.796417	0.668031	0.020155	0.814592	0.681633
4	3	59.04	0.798081	0.693841	0.026416	0.818795	0.696698
5	4	60.27	0.763950	0.697565	0.028528	0.836244	0.702075
6	123456789	68.20	0.504766	0.319372	0.612469	0.482819	0.358713
7	1111111	57.14	0.845179	0.293034	0.555058	0.176051	0.897143
8	6999	55.06	0.888627	0.265917	0.437478	0.108680	0.346341
9	65536	82.50	0.127817	0.752702	0.834128	0.448534	0.171280
10	16777216	78.86	0.195292	0.835838	0.740581	0.506603	0.077628
11	1078741824	70.28	0.434553	0.170393	0.378881	0.237267	0.535928
12	10	63.30	0.670740	0.647165	0.042954	0.876302	0.727100
13	100	66.78	0.553262	0.580088	0.265777	0.885728	0.491686
14	194305786	69.32	0.466697	0.786260	0.757810	0.034240	0.008054
15	1217344457	68.30	0.501262	0.099260	0.006290	0.744610	0.710015
16	314159276	66.24	0.571856	0.060258	0.100924	0.874196	0.541280
17	543219876	77.37	0.229110	0.674521	0.165676	0.432569	0.556232
18	9571916	63.84	0.653023	0.034591	0.207023	0.126443	0.600962
19	5868958	80.80	0.156785	0.619371	0.000764	0.912909	0.359614
20	1234567890	91.74	0.035146	0.042005	0.043784	0.874018	0.699561
21	1933985544	80.93	0.154323	0.740990	0.415336	0.851022	0.073228
22	2050954260	80.04	0.171185	0.959658	0.839948	0.529940	0.023087
23	918807827	71.46	0.395971	0.713819	0.755923	0.561918	0.763137

RUN CHI-SQUARE TEST FOR GENERATOR IN GSS
25000 RUNS OF INCREASING VALUES TO LENGTH 6

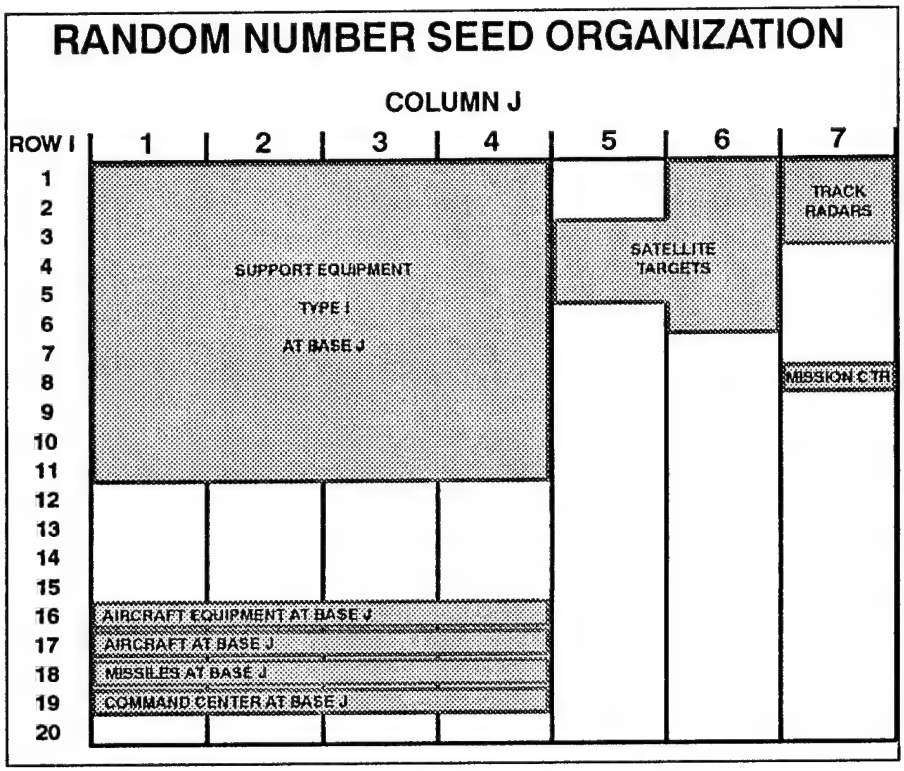
SEED	VALUE	CHISQUARE	P VALUE
1	0	2.62	0.758752
2	1	3.01	0.699083
3	2	3.12	0.680722
4	3	3.14	0.678316
5	4	3.03	0.696071
6	123456789	1.45	0.918415
7	1111111	4.76	0.445398
8	6999	8.49	0.131299
9	65536	9.39	0.094325
10	16777216	14.19	0.014461
11	1078741824	2.03	0.844532
12	10	3.10	0.684632
13	100	2.82	0.727664
14	194305786	4.28	0.509403
15	1217344457	2.64	0.755481
16	314159276	2.03	0.845620
17	543219876	1.57	0.905049
18	9571916	8.92	0.112141
19	5868958	2.11	0.833536
20	1234567890	2.15	0.827968
21	1933985544	9.89	0.078337
22	2050954260	5.50	0.357518
23	918807827	4.67	0.457222

OVERLAPPING TRIPLES TEST FOR GSS
30000 SETS, SORTED INTO 8 BY 8 BY 8 GRID

SEED	VALUE	CHISQUARE	P VALUE
1	0	468.85	0.240331
2	1	473.13	0.199070
3	2	471.31	0.216072
4	3	469.83	0.230480
5	4	470.36	0.225226
6	123456789	487.82	0.093633
7	1111111	477.75	0.159984
8	6999	465.36	0.277197
9	65536	459.91	0.340144
10	16777216	476.95	0.166347
11	1078741824	465.72	0.273229
12	10	476.45	0.170379
13	100	477.82	0.159381
14	194305786	501.83	0.038867
15	1217344457	464.19	0.290232
16	314159276	460.35	0.334903
17	543219876	467.99	0.249178
18	9571916	456.84	0.378049
19	5868958	478.17	0.156650
20	1234567890	500.81	0.041656
21	1933985544	499.03	0.046930
22	2050954260	450.51	0.460012
23	918807827	471.16	0.217500

APPENDIX D - RANDOM NUMBER SYNCHRONIZATION IN OSPREY

Osprey is a space defense mission effectiveness model developed by Teledyne Brown Engineering and used in support of many antisatellite system studies by the Army Kinetic Energy ASAT System Program Office, Air Force Operational Test and Evaluation Center, Space Command, and various contractors. The random number synchronization scheme was developed by Jeff Niemuth of Teledyne Brown in 1978. The material here is from the Analyst Manual written for the version that supported the ASAT Initial Operational Test and Evaluation [Webb 1987].



The entities for which random draws are needed are interceptor missiles, aircraft used to deliver the missiles to their designated launch point and launch them, carrier aircraft equipment, 11 types of ground support equipment, base control center, satellite targets, ground based satellite tracking radars, and a single overall mission control center. All but the last three are associated with the particular base at which they are located. The generators are organized into a two-dimensional array as shown.

The actual random number seeds are stored in a singly dimensioned array JRANGS. When a random number is needed for entity K associated with row I and column J of the organization scheme, an index is constructed by the formula

$$\text{INDEX} = \text{IBLOCK}(I, J) - \text{IBEGIN}(I, J) + K.$$

The value of INDEX is used to select the element in JRANGS to use as the seed for the particular draw. Here IBLOCK is an array of starting points in JRANGS and IBEGIN is an array of first entity numbers for block I J. Most entries in IBEGIN are 1, except for some entities that are numbered consecutively across bases. Thus IBLOCK(I, J) is the seed for the first entity in position I J. The values used in IBLOCK and the corresponding number of seeds associated with each block (equivalent to the maximum number of entities of a class) are as follows:

Row I	Column J						
	1	2	3	4	5	6	7
1	1	21	41	61	0	1461	1881
2	81	101	121	141	0	1531	1961
3	161	181	201	221	1251	1601	1921
4	241	261	281	301	1321	1621	0
5	321	341	361	381	1391	1741	0
6	401	421	441	461	0	1811	0
7	481	501	521	541	0	0	0
8	561	581	601	621	0	0	649
9	1011	1031	1051	1071	0	0	0
10	1091	1111	1131	1151	0	0	0
11	1171	1191	1211	1231	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0
16	645	646	647	648	0	0	0
17	651	676	701	726	0	0	0
18	751	816	881	946	0	0	0
19	641	642	643	644	0	0	0
20	0	0	0	0	0	0	0

Row I	Column J						
	1	2	3	4	5	6	7
1	20	20	20	20	-	70	40
2	20	20	20	20	-	70	40
3	20	20	20	20	70	70	40
4	20	20	20	20	70	70	-
5	20	20	20	20	70	70	-
6	20	20	20	20	-	70	-
7	20	20	20	20	-	-	-
8	20	20	20	20	-	-	2
9	20	20	20	20	-	-	-
10	20	20	20	20	-	-	-
11	20	20	20	20	-	-	-
16	1	1	1	1	-	-	-
17	25	25	25	25	-	-	-
18	65	65	65	65	-	-	-
19	1	1	1	1	-	-	-

The advantage of this scheme is that it will accommodate irregular indexing structures for different entities and different quantities of different entities efficiently. As the model changes, it is possible to restructure the scheme by changing the values in IBLOCK and adjusting the indices used in each random number call accordingly.

APPENDIX E – EXPERIMENT DESIGNS FOR SIMULATION STUDIES

Designs for 3 through 10 factors at 3 levels are given. The levels are denoted as 0 = low, 1 = nominal, and 2 = high. Efficiencies are given that compare the average variance to that obtainable with the full factorial, on a per run basis.

Three factors. 96%

1) 1 1 1
2) 1 1 0
3) 1 0 1
4) 0 1 1
5) 1 1 2
6) 1 2 1
7) 2 1 1
8) 0 0 0
9) 0 2 2
10) 2 0 2
11) 2 2 0
12) 0 0 2
13) 2 2 2
14) 0 2 0
15) 2 0 0

Four factors. 82%

1) 1 1 1 1
2) 1 1 1 0
3) 1 1 0 1
4) 1 0 1 1
5) 0 1 1 1
6) 1 1 1 2
7) 1 1 2 1
8) 1 2 1 1
9) 2 1 1 1
10) 1 0 0 0
11) 1 0 2 2
12) 1 2 0 2
13) 1 2 2 0
14) 0 0 0 2
15) 0 0 2 0
16) 0 2 0 0
17) 2 2 2 2
18) 2 2 0 0
19) 2 0 0 2
20) 0 2 2 2
21) 2 0 2 0

Five factors. 81%

1) 1 1 1 1 1
2) 1 1 1 1 0
3) 1 1 1 0 1
4) 1 1 0 1 1
5) 1 0 1 1 1
6) 0 1 1 1 1
7) 1 1 1 1 2
8) 1 1 1 2 1
9) 1 1 2 1 1
10) 1 2 1 1 1
11) 2 1 1 1 1
12) 0 0 2 2 2
13) 0 2 0 2 2
14) 0 2 2 0 2
15) 0 2 2 2 0
16) 2 0 0 2 2
17) 2 0 2 0 2
18) 2 0 2 2 0
19) 2 2 0 0 2
20) 2 2 0 2 0
21) 2 2 2 0 0
22) 2 2 2 2 2
23) 0 0 2 0 0
24) 0 2 0 0 0
25) 2 0 0 0 0
26) 0 0 0 0 2
27) 0 0 0 2 0

Six factors. 70%

1) 1 1 1 1 1 1
2) 1 1 1 1 1 0
3) 1 1 1 1 0 1
4) 1 1 1 0 1 1
5) 1 1 0 1 1 1
6) 1 0 1 1 1 1
7) 0 1 1 1 1 1
8) 1 1 1 1 1 2
9) 1 1 1 1 2 1
10) 1 1 1 2 1 1
11) 1 1 2 1 1 1
12) 1 2 1 1 1 1
13) 2 1 1 1 1 1
14) 0 0 0 0 0 2
15) 0 0 0 0 2 0
16) 0 0 0 2 0 0
17) 0 0 2 0 0 0
18) 0 2 0 0 0 0
19) 2 0 0 0 0 0
20) 0 0 2 2 2 2
21) 0 2 0 2 2 2
22) 0 2 2 0 2 2
23) 0 2 2 2 0 2
24) 0 2 2 2 2 0

25) 2 0 0 2 2 2
26) 2 0 2 0 2 2
27) 2 0 2 2 0 2
28) 2 0 2 2 2 0
29) 2 2 0 0 2 2
30) 2 2 0 2 0 2
31) 2 2 0 2 2 0
32) 2 2 2 0 0 2
33) 2 2 2 0 2 0
34) 2 2 2 2 0 0
35) 2 2 2 2 2 2

Seven factors. 59%

1) 1 1 1 1 1 1 1
2) 1 1 1 1 1 1 0
3) 1 1 1 1 1 0 1
4) 1 1 1 1 0 1 1
5) 1 1 1 0 1 1 1
6) 1 1 0 1 1 1 1
7) 1 0 1 1 1 1 1
8) 0 1 1 1 1 1 1
9) 1 1 1 1 1 1 2
10) 1 1 1 1 1 2 1
11) 1 1 1 1 2 1 1
12) 1 1 1 2 1 1 1
13) 1 1 2 1 1 1 1
14) 1 2 1 1 1 1 1
15) 2 1 1 1 1 1 1
16) 0 0 0 0 0 0 0
17) 2 2 0 0 0 0 0
18) 2 0 2 0 0 0 0
19) 2 0 0 2 0 0 0
20) 2 0 0 0 2 0 0
21) 2 0 0 0 0 2 0
22) 2 0 0 0 0 0 2
23) 0 2 2 0 0 0 0
24) 0 2 0 2 0 0 0
25) 0 2 0 0 2 0 0
26) 0 2 0 0 0 2 0
27) 0 2 0 0 0 0 2
28) 0 0 2 2 0 0 0
29) 0 0 2 0 2 0 0
30) 0 0 2 0 0 2 0
31) 0 0 2 0 0 0 2
32) 0 0 0 2 2 0 0
33) 0 0 0 2 0 2 0
34) 0 0 0 2 0 0 2
35) 0 0 0 0 2 2 0
36) 0 0 0 0 2 0 2
37) 0 0 0 0 0 2 2
38) 0 2 2 2 2 2 2
39) 2 0 2 2 2 2 2
40) 2 2 0 2 2 2 2
41) 2 2 2 0 2 2 2
42) 2 2 2 2 0 2 2
43) 2 2 2 2 2 0 2
44) 2 2 2 2 2 2 0

Eight factors. 53%

1) 1 1 1 1 1 1 1 1
2) 1 1 1 1 1 1 1 0
3) 1 1 1 1 1 1 0 1
4) 1 1 1 1 1 0 1 1
5) 1 1 1 1 0 1 1 1
6) 1 1 1 0 1 1 1 1
7) 1 1 0 1 1 1 1 1
8) 1 0 1 1 1 1 1 1
9) 0 1 1 1 1 1 1 1
10) 1 1 1 1 1 1 1 2
11) 1 1 1 1 1 1 2 1
12) 1 1 1 1 1 2 1 1
13) 1 1 1 1 2 1 1 1
14) 1 1 1 2 1 1 1 1
15) 1 1 2 1 1 1 1 1
16) 1 2 1 1 1 1 1 1
17) 2 1 1 1 1 1 1 1
18) 0 0 0 2 0 0 0 2
19) 0 0 0 2 0 0 2 0
20) 0 0 0 2 0 2 0 0
21) 0 0 0 2 2 0 0 0
22) 0 0 2 0 0 0 0 2
23) 0 0 2 0 0 0 2 0
24) 0 0 2 0 0 2 0 0
25) 0 0 2 0 2 0 0 0
26) 0 2 0 0 0 0 0 2
27) 0 2 0 0 0 0 2 0
28) 0 2 0 0 0 2 0 0
29) 0 2 0 0 2 0 0 0
30) 2 0 0 0 0 0 0 2
31) 2 0 0 0 0 0 2 0
32) 2 0 0 0 0 2 0 0
33) 2 0 0 0 2 0 0 0
34) 0 0 2 2 2 2 2 2
35) 0 2 0 2 2 2 2 2
36) 0 2 2 0 2 2 2 2
37) 2 0 0 2 2 2 2 2
38) 2 0 2 0 2 2 2 2
39) 2 2 0 0 2 2 2 2
40) 0 2 2 2 0 0 0 0
41) 2 0 2 2 0 0 0 0
42) 2 2 0 2 0 0 0 0
43) 2 2 2 0 0 0 0 0
44) 2 2 2 2 0 0 2 2
45) 2 2 2 2 0 2 0 2
46) 2 2 2 2 0 2 2 0
47) 2 2 2 2 2 0 0 2
48) 2 2 2 2 2 0 2 0
49) 2 2 2 2 2 2 0 0

50) 0 0 0 0 0 0 0 0 0
51) 0 0 0 0 0 0 2 2 2
52) 0 0 0 0 2 2 2 0
53) 0 0 0 1 2 0 2 2
54) 0 0 0 1 2 2 0 2

Nine factors. 46%

1) 1 1 1 1 1 1 1 1 1
2) 1 1 1 1 1 1 1 1 0
3) 1 1 1 1 1 1 1 0 1
4) 1 1 1 1 1 1 0 1 1
5) 1 1 1 1 1 0 1 1 1
6) 1 1 1 1 0 1 1 1 1
7) 1 1 1 0 1 1 1 1 1
8) 1 1 0 1 1 1 1 1 1
9) 1 0 1 1 1 1 1 1 1
10) 0 1 1 1 1 1 1 1 1
11) 1 1 1 1 1 1 1 1 2
12) 1 1 1 1 1 1 1 2 1
13) 1 1 1 1 1 1 2 1 1
14) 1 1 1 1 1 2 1 1 1
15) 1 1 1 1 2 1 1 1 1
16) 1 1 1 2 1 1 1 1 1
17) 1 1 2 1 1 1 1 1 1
18) 1 2 1 1 1 1 1 1 1
19) 2 1 1 1 1 1 1 1 1
20) 0 0 0 2 0 0 0 0 2
21) 0 0 0 2 0 0 0 2 0
22) 0 0 0 2 0 0 2 0 0
23) 0 0 0 2 0 2 0 0 0
24) 0 0 0 2 2 0 0 0 0
25) 0 0 2 0 0 0 0 0 2
26) 0 0 2 0 0 0 0 2 0
27) 0 0 2 0 0 0 2 0 0
28) 0 0 2 0 0 2 0 0 0
29) 0 0 2 0 2 0 0 0 0
30) 0 2 0 0 0 0 0 0 2
31) 0 2 0 0 0 0 0 2 0
32) 0 2 0 0 0 0 2 0 0
33) 0 2 0 0 0 2 0 0 0
34) 0 2 0 0 2 0 0 0 0
35) 2 0 0 0 0 0 0 0 2
36) 2 0 0 0 0 0 0 2 0
37) 2 0 0 0 0 0 2 0 0
38) 2 0 0 0 0 2 0 0 0
39) 2 0 0 0 2 0 0 0 0
40) 0 0 2 2 2 2 2 2 2
41) 0 2 0 2 2 2 2 2 2
42) 0 2 2 0 2 2 2 2 2
43) 2 0 0 2 2 2 2 2 2
44) 2 0 2 0 2 2 2 2 2
45) 2 2 0 0 2 2 2 2 2
46) 2 2 2 2 0 0 2 2 2
47) 2 2 2 2 0 2 0 2 2
48) 2 2 2 2 0 2 2 0 2
49) 2 2 2 2 0 2 2 2 0
50) 2 2 2 2 2 0 0 2 2
51) 2 2 2 2 2 0 2 0 2
52) 2 2 2 2 2 0 2 2 0
53) 2 2 2 2 2 2 0 0 2
54) 2 2 2 2 2 2 0 2 0
55) 2 2 2 2 2 2 2 0 0
56) 2 2 2 2 2 2 2 2 2
57) 0 0 0 0 2 2 2 2 0
58) 0 0 0 0 2 2 0 2 2
59) 0 0 0 0 2 0 2 2 2
60) 0 0 0 0 0 2 2 2 2
61) 0 0 0 0 2 2 2 0 2
62) 2 2 2 0 0 0 0 0 0
63) 2 0 2 2 0 0 0 0 0
64) 2 2 0 2 0 0 0 0 0
65) 0 2 2 2 0 0 0 0 0

Ten factors. 41%

1) 1 1 1 1 1 1 1 1 1 1
2) 1 1 1 1 1 1 1 1 1 0
3) 1 1 1 1 1 1 1 1 0 1
4) 1 1 1 1 1 1 1 0 1 1
5) 1 1 1 1 1 1 0 1 1 1
6) 1 1 1 1 1 0 1 1 1 1
7) 1 1 1 1 0 1 1 1 1 1
8) 1 1 1 0 1 1 1 1 1 1
9) 1 1 0 1 1 1 1 1 1 1
10) 1 0 1 1 1 1 1 1 1 1
11) 0 1 1 1 1 1 1 1 1 1
12) 1 1 1 1 1 1 1 1 1 2
13) 1 1 1 1 1 1 1 1 2 1
14) 1 1 1 1 1 1 1 2 1 1
15) 1 1 1 1 1 1 2 1 1 1
16) 1 1 1 1 1 2 1 1 1 1
17) 1 1 1 1 2 1 1 1 1 1
18) 1 1 1 2 1 1 1 1 1 1
19) 1 1 2 1 1 1 1 1 1 1
20) 1 2 1 1 1 1 1 1 1 1
21) 2 1 1 1 1 1 1 1 1 1
22) 0 0 0 0 0 0 0 0 0 0
23) 0 0 0 0 2 0 0 0 0 2
24) 0 0 0 0 2 0 0 0 2 0
25) 0 0 0 0 2 0 0 2 0 0
26) 0 0 0 0 2 0 2 0 0 0
27) 0 0 0 0 2 2 0 0 0 0
28) 0 0 0 2 0 0 0 0 0 2
29) 0 0 0 2 0 0 0 0 2 0
30) 0 0 0 2 0 0 0 2 0 0
31) 0 0 0 2 0 0 2 0 0 0
32) 0 0 0 2 0 2 0 0 0 0
33) 0 0 2 0 0 0 0 0 0 2
34) 0 0 2 0 0 0 0 0 2 0

35) 0 0 2 0 0 0 0 2 0 0
36) 0 0 2 0 0 0 2 0 0 0
37) 0 0 2 0 0 2 0 0 0 0
38) 0 2 0 0 0 0 0 0 0 2
39) 0 2 0 0 0 0 0 0 2 0
40) 0 2 0 0 0 0 0 2 0 0
41) 0 2 0 0 0 0 2 0 0 0
42) 0 2 0 0 0 2 0 0 0 0
43) 2 0 0 0 0 0 0 0 0 2
44) 2 0 0 0 0 0 0 0 2 0
45) 2 0 0 0 0 0 0 2 0 0
46) 2 0 0 0 0 0 2 0 0 0
47) 2 0 0 0 0 2 0 0 0 0
48) 0 0 2 2 2 2 2 2 2 2
49) 0 2 0 2 2 2 2 2 2 2
50) 0 2 2 0 2 2 2 2 2 2
51) 0 2 2 2 0 2 2 2 2 2
52) 2 0 0 2 2 2 2 2 2 2
53) 2 0 2 0 2 2 2 2 2 2
54) 2 0 2 2 0 2 2 2 2 2
55) 2 2 0 0 2 2 2 2 2 2
56) 2 2 0 2 0 2 2 2 2 2
57) 2 2 2 0 0 2 2 2 2 2
58) 2 2 2 2 2 0 0 2 2 2
59) 2 2 2 2 2 0 2 0 2 2
60) 2 2 2 2 2 0 2 2 0 2
61) 2 2 2 2 2 0 2 2 2 0
62) 2 2 2 2 2 2 0 0 2 2
63) 2 2 2 2 2 2 0 2 0 2
64) 2 2 2 2 2 2 0 2 2 0
65) 2 2 2 2 2 2 2 0 0 2
66) 2 2 2 2 2 2 2 0 2 0
67) 2 2 2 2 2 2 2 2 0 0
68) 0 0 0 0 0 0 2 2 2 2
69) 0 0 0 0 0 2 0 2 2 2
70) 0 0 0 0 0 2 2 0 2 2
71) 0 0 0 0 0 2 2 2 0 2
72) 0 0 0 0 0 2 2 2 2 0
73) 0 2 2 2 2 0 0 0 0 0
74) 2 0 2 2 2 0 0 0 0 0
75) 2 2 0 2 2 0 0 0 0 0
76) 2 2 2 0 2 0 0 0 0 0
77) 2 2 2 2 0 0 0 0 0 0

APPENDIX F – EXPERIMENT RESULTS

This appendix gives details of the experiment involving the phone-line simulation discussed on page 36. The responses are average number of customers served in a day, maximum queue length, server utilization percentage, and average waiting time. Seven factors are varied in the experiment.

ORIGINAL DESIGN AND OBSERVATIONS

1)	1	1	1	1	1	1	1	146.18	5.34	55.21	1.68
2)	0	1	1	1	1	1	1	145.19	4.80	54.78	1.40
3)	2	1	1	1	1	1	1	146.53	5.58	55.37	1.87
4)	1	0	1	1	1	1	1	146.80	4.94	52.46	1.12
5)	1	2	1	1	1	1	1	145.60	5.67	57.14	2.21
6)	1	1	0	1	1	1	1	146.16	5.18	53.23	1.57
7)	1	1	2	1	1	1	1	145.95	6.35	57.26	2.29
8)	1	1	1	0	1	1	1	142.11	7.14	68.26	4.94
9)	1	1	1	2	1	1	1	146.89	3.97	45.57	.61
10)	1	1	1	1	0	1	1	146.61	4.50	49.60	1.09
11)	1	1	1	1	2	1	1	145.39	6.01	60.65	2.48
12)	1	1	1	1	1	0	1	142.99	5.34	56.13	1.71
13)	1	1	1	1	1	2	1	149.65	5.34	54.46	1.65
14)	1	1	1	1	1	1	0	153.60	5.75	57.92	1.99
15)	1	1	1	1	1	1	2	139.41	4.72	52.69	1.42
16)	0	0	0	0	0	0	0	148.37	5.59	59.53	2.05
17)	2	2	0	0	0	0	0	147.32	7.71	66.64	5.78
18)	2	0	2	0	0	0	0	150.94	7.64	65.48	3.62
19)	0	2	2	0	0	0	0	142.81	6.35	69.42	4.50
20)	2	0	0	2	0	0	0	151.87	2.99	40.32	.24
21)	0	2	0	2	0	0	0	151.31	3.29	43.32	.45
22)	0	0	2	2	0	0	0	150.96	5.23	43.24	.85
23)	2	0	0	0	2	0	0	147.46	9.15	73.03	6.85
24)	0	2	0	0	2	0	0	135.27	6.79	74.65	6.61
25)	0	0	2	0	2	0	0	141.76	6.73	75.70	5.19
26)	0	0	0	2	2	0	0	151.01	4.02	49.51	.61
27)	2	0	0	0	0	2	0	157.46	6.85	58.86	2.77
28)	0	2	0	0	0	2	0	150.06	5.98	62.99	3.64
29)	0	0	2	0	0	2	0	154.20	6.05	61.85	2.67
30)	0	0	0	2	0	2	0	158.12	2.82	39.08	.20
31)	0	0	0	0	2	2	0	148.70	6.50	68.52	4.16
32)	2	0	0	0	0	0	2	136.76	5.63	54.98	1.91
33)	0	2	0	0	0	0	2	131.94	5.38	59.77	3.06
34)	0	0	2	0	0	0	2	135.05	5.58	58.57	2.24
35)	0	0	0	2	0	0	2	136.90	2.23	36.34	.12
36)	0	0	0	0	2	0	2	131.65	5.94	65.20	3.45
37)	0	0	0	0	0	2	2	141.46	4.85	52.98	1.47
38)	0	2	2	2	2	2	2	141.57	5.25	50.11	1.36
39)	2	0	2	2	2	2	2	142.85	5.73	46.89	1.09
40)	2	2	0	2	2	2	2	142.77	4.09	47.10	.88
41)	2	2	2	0	2	2	2	136.35	8.90	75.91	9.96
42)	2	2	2	2	0	2	2	142.86	5.60	40.98	.98
43)	2	2	2	2	2	0	2	136.86	6.09	52.32	1.64
44)	2	2	2	2	2	2	0	157.73	6.93	55.80	2.13

PARAMETER ESTIMATES

mean	144.8671	5.6006	55.9494	2.5133
A	1.0669	.5848	.5123	.4581
Asq	-.1155	-.0425	-.0563	-.0250
B	-1.1394	.2493	2.2434	.6965
Bsq	-.0022	-.0042	-.1470	-.0152
C	-.1188	.6654	2.0335	.4311
Csq	-.0505	.1491	.0012	.0735
D	2.5898	-1.3193	-11.0515	-2.2849
Dsq	-.5689	.0791	.5574	.3560
E	-1.3032	.5143	5.2345	.8880
Esq	-.0689	-.0209	-.0396	.0259
F	3.1500	.0145	-.8183	.0312
Fsq	.0378	.0075	.0187	-.0094
G	-6.8271	-.3721	-2.5081	-.2271
Gsq	.0995	-.0275	.0211	-.0029
AB	.2850	.0980	.1884	.1842
AC	.1236	.0500	.0394	-.0311
AD	-.7741	-.3535	-.4016	-.4026
AE	.5005	.2412	.2943	.2820
AF	-.0169	.0116	-.0666	-.0320
AG	-.4883	-.1329	-.2288	-.1099
BC	.0945	-.0183	.1392	.0277
BD	1.0893	-.0243	-.5379	-.5126
BE	-.5436	-.0945	-.1882	.0598
BF	.1365	.0484	.0794	.0253
BG	.3101	.0589	.0837	.0163
CD	-.0645	.4301	-.4135	-.0864
CE	.0125	-.1401	.2184	.0294
CF	.1051	.0353	-.0624	.1076
CG	.0213	.0783	-.0739	.1355
DE	1.1948	-.0586	-.4821	-.6436
DF	-.0451	-.0132	.0944	-.0359
DG	-.6464	-.0002	.2173	.0799
EF	-.0230	-.0384	-.0307	-.0142
EG	.4181	-.0179	-.0512	-.0576
FG	-.1468	.0549	.0758	.1554

APPENDIX G – NETWORK MODEL RESULTS

The technique of staged aggregation was proposed as a candidate for speeding up large-scale simulation models in which there are very many essentially identical elements, such as for simulating traffic flow in very large networks. Some work was done on a network model to attempt to develop the concept. This appendix summarizes the results, which were essentially all negative. The work was abandoned when it appeared that the technique was inappropriate for engineering simulations of interest to CECOM.

The model developed is an idealized model that will be referred to as the rumor model. N^2 nodes are connected in a two-dimensional Cartesian grid of side N. At some epoch an event occurs that causes one of the nodes to contact its four neighbors in a random order (presumably to tell them about the event). After each node is contacted, it contacts its neighbors, again in a random order, to pass along the message. The simulation of this system is constructed to loop through the nodes to see which have received the message but have not yet contacted all four neighbors, then effect one of these contacts. The response variables are 1) the number of iterations of the loop before all nodes have been notified, 2) the number of calls "wasted" in the sense that they are made to a node that has already received the message, and 3) the number of calls made out of the grid to phantom neighbors located outside the perimeter of the grid; separate counts are made in each of the four directions, so this represents four outputs.

Rather than simulating the system by modeling each node, it could also be simulated by grouping nodes together and modeling the behavior of the group. The behavior of the group is more complex, but there are fewer groups than there are nodes. If N = 16, then the system contains 256 nodes. If these are simulated in groups of 4 (a 2 × 2 grid), then only 64 groups must be simulated. Alternatively, they could be simulated in groups of 16 as a 4 × 4 grid, and the behavior of the 4 × 4 grid could be determined by simulating 4 groups of 2 × 2 grids.

The principle of staged aggregation works with a sequence of groupings of the individual elements as described here. If the complexity of the models for the groupings does not grow at too great a rate, and if the rules for combining the models do not increase in complexity, then there could be a net saving in simulation effort by doing the staged aggregation over simulating the elements individually.

Several variants of this basic model were developed. The first set looked at aggregation of the model into blocks: (1) a 16 × 16 array of nodes was compared with (2) an 8 × 8 array of blocks of 2 × 2 nodes; (3) a 4 × 4 array of blocks consisting of 2 × 2 sub-blocks of 2 × 2 nodes; and (4) a 2 × 2 array of blocks consisting of 2 × 2 sub-blocks, which in turn consist of 2 × 2 sub-sub-blocks of 2 × 2 nodes. Delays were introduced to represent the time taken in identifying which nodes will make and receive the calls, setting up the calls, and passing on the message.

A second set looked at an alternate stopping rule: the run ended when the message was received at the opposite corner from which it started. In this case an

additional response is how many nodes actually received the message.

A third set replaced the loop structure with an event calendar. When a message was received at a node, an event was scheduled at the next time step to make calls to pass on the message. An added feature was the use of a call list; if a call was made to a node that was already on the list for a given time step, then a busy signal was received and the call must be repeated later. This variant was combined with the four levels of aggregation into blocks.

A refinement of the event structure was to add a feature wherein a node would act as a sink with a certain probability. An output statistic here was the number of nodes that received the message before it died because no active node would pass it on.

Run times were recorded for these cases. The delay incorporated into the models has been adjusted so that the runs times for 100 Monte Carlo samples lie in the range 5 to 10 minutes on a 486 processor running at 33 MHz. It has been found that the more complex indexing schemes of the aggregated models add only a few seconds time penalty.

Results were largely negative; no ways of capitalizing on the variations in model structure were found. The sole exception is a variant of the 2-D loop model in which the results of the first stimulation of a block of 4 nodes has been precomputed and is supplied by a special subroutine. For this case a very modest time saving of only about 10% is seen.

A further variant considers a message started at the center of the grid. As a given node receives the message, it may serve as a sink by failing to pass the message along with a given probability structure. The parameters of the problem are set so that the message dies out before all the nodes of the complete grid receive the message.

If, on the other hand, the message starts at a corner of the grid, say the lower left corner, then the pattern of message transmittal will be similar to a quarter of the pattern if the message starts in the center. It will not be the same, however, because messages may be sent outside the boundaries of the grid to the left and down, but are not received from those directions. The technique studied attempts to rectify the quarter-plane model by the addition of a new event type that introduces messages received along the borders. These represent messages that in the full plane would have been propagated by going into an adjacent quadrant then being passed back. Initially there is no information on which to build a model for such reintroduction. The idea used is to estimate the probability of the occurrence of a reintroduction event from the model with no reintroduction, then iteratively correct the probabilities until stability is achieved. The details will now be described as implemented in a test example.

Let the full grid be of size 31×31 nodes. Let the nodes be numbered (x,y), where x and y can have the integer values -15 to $+15$. The starting point for the initial message is (0,0). This simulation model is replicated 100 times. This will be considered the reference case.

The reference case is to be compared with a modified simulation of a 16 × 16 grid, with nodes numbered with the integer values 0 to +15. The starting point is again (0,0). The objective is to see if the reference case can be duplicated approximately by a scaled version of the smaller modified case. Consider a node (i,0) on the lower border of the array in the smaller case. This node will send messages out of the array to node (i,−1), but cannot receive messages from that node because it is not modeled in the simulation. The model is instrumented by recording the number of occurrences in which a node (i,1) sends a message to node (i,0), and the simulation time of each occurrence. This number of occurrences in 100 samples is used to estimate the probability that a message should have been received by node (i,0) from the missing node (i,−1).

The next step is to modify the model to have messages introduced by the bordering nodes. The probability that a node (i,0) will receive such a message is a new input, as is a linear expression for the simulation time at which the input would occur. By the symmetry of the problem, the same expressions are used for introducing messages from (−1,j) to (0,j). The revised simulation is run to obtain improved estimates of these probabilities and the expression for simulation time of occurrence. It was found that the second estimates differed slightly from the initial ones, but that further iterations were stable. That is, the third run, using the second estimate of probabilities and time of occurrence, gives a revised estimate that does not differ significantly from the second estimate.

The models were rigged with artificial time delays representing computation that would occur in a real model of a communication system. The time taken by the reference case was 1423.0 seconds on the 486 PC. The time taken by one run of the smaller case was 406.3 seconds. Depending on assumptions made about the effort required to establish the parameters for the smaller model, there may or may not be a time saving.

REFERENCES

Abramowitz, M. and I. Stegun [1964]. *Handbook of mathematical functions.* US Department of Commerce, National Bureau of Standards, Applied Mathematics series 55.

Army, Department of [1986]. JANUS (T) documentation. US Army TRADOC Analysis Center.

Bratley, Paul, Bennett L. Fox, and Linus E. Schrage [1983]. *A guide to simulation.* Springer-Verlag.

ECAC [1983]. The terrain integrated rough earth model (TIREM) technical note. Electromagnetic Compatibility Analysis Center technical note ECAC-TN-83-002.

Joint Staff [1992]. Catalog of wargaming and military simulation models, 12th edition. Force Structure, Resource, and Assessment Directorate (J-8), The Joint Staff, Washington, DC. DTIC AD-A246 431.

Knuth, Donald E. [1969]. *The art of computer programming: Volume 2 – Seminumerical algorithms.* Addison-Wesley.

Kruger, Anton [1990]. *Efficient Fortran programming.* Wiley.

Marsaglia, George [1985]. A current view of random number generators. *Computer science and statistics: The interface*, L. Billard, ed. Elsevier Science Publishers.

Mood, Alexander M. [1950]. *Introduction to the theory of statistics.* McGraw-Hill.

Ritchie, Adelia E., ed [1992]. Simulation validation workshop proceedings. Military Operations Research Society. DTIC AD-A276 941.

Sauerborn, Geoffrey C. [1995]. Distributed interactive simulation (DIS) protocol data units (PDUs) implemented into a combat model. Army Research Laboratory ARL-MR-227.

Schuppe, Thomas F. [1991]. Modeling and simulation: a Department of Defense critical technology. *Proc 1991 winter simulation conference*, pp 519-524.

Webb, Steve et al [1987]. Analyst manual for the Osprey ASAT system effectiveness model. Teledyne Brown Engineering Report MK11-87-AFOTEC-70.

Wray, John W. [1997]. Validation of CECOM's System Performance Model. Unpublished briefing charts dated September 17.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	31 Oct 97	Final Report 16 Dec 96 - 30 Sep 97

4. TITLE AND SUBTITLE
MODELING AND SIMULATION TECHNIQUES FOR LARGE-SCALE COMMUNICATIONS MODELING

5.FUNDING NUMBERS
C: DAAB07-97-C-D256

6. AUTHOR(S)
Steve Webb

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Steve Webb
809 Grayling Bay
Costa Mesa, CA 92626

8. PERFORMING ORGANIZATION REPORT NUMBER
SW-97-11

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
US Army CECOM
Fort Monmouth, NJ 07703-5008

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release. Distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Because modeling and simulation are so widely used throughout the Army, vast amounts of computer time are used. Of more concern, however, is the quantity of analyst time involved in setting up and analyzing the results of these runs. Increased confidence in the models may be expected to lead to more efficient use of the analyst's time. To this end, an ongoing validation effort for a CECOM model was supported by providing computer routines that measure the degree to which simulation data match test data. Tests of random number generators were also developed and applied to CECOM models. It was found that synchronization of random number strings in simulations is easy to implement and can provide significant savings for making comparative studies. If synchronization is in place, then statistical experiment design can be used to provide information on the sensitivity of the output to input parameters. The report concludes with recommendations and an implementation plan.

14. SUBJECT TERMS
Communications modeling; Simulation validation; Random number generators; Random number testing; Random number synchronization; Statistical experiment design; Modeling and simulation

15. NUMBER OF PAGES
87 + iv

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	